**Titulní list**







# ASEMBLERY
# IAS

## Studijní opora

**Tento učební text vznikl za podpory projektu „Zvýšení konkurenceschopnosti IT odborníků – absolventů pro Evropský trh práce“, reg. č. CZ.04.1.03/3.2.15.1/0003. Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.**

doc. Dr. Ing. Petr Hanáček
doc. Ing. František Zbořil, CSc.

Verze: 2.2006

## Obsah

# 1 PŘEDMLUVA

**1.1 Organizační informace**



Tato studijní opora je pomocným učebním textem pro stejnojmenný předmět, pokrývá přibližně 80 % obsahu předmětu a je zdrojem minimálního rozsahu znalostí potřebných pro úspěšné absolvování tohoto předmětu. Její text by měl být skutečnou „oporou“, to znamená, že by měl pouze doplňovat ostatní studijní materiály předmětu, a proto tato studijní opora nenahrazuje knižní učebnici.

Předmět se skládá z přednášek a z vedených počítačových cvičení. Účast na přednáškách není povinná. Zkušenosti však přesvědčivě ukazují, že neúčast na přednáškách patří mezi nejčastější příčiny neúspěchu při půlsemestrální i závěrečné zkoušce.

Podrobné informace o předmětu, s aktualizacemi pro každý akademický rok, lze nalézt na adrese http://www.fit.vutbr.cz/study/courses/IAS/private/, ke které mají přístup všichni zapsaní studenti.

**Zkouška a hodnocení předmětu**



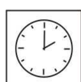
V počítačových cvičeních lze za předepsané aktivity získat až 20 bodů. Dalších 20 bodů lze získat při půlsemestrální písemné zkoušce. Podmínkou zápočtu je pak získání minimálně 15 bodů (z výše uvedených 40 možných bodů). Neudělení zápočtu znamená neúspěšné absolvování předmětu.

Závěrečnou písemnou zkoušku (max. 60 bodů) smí psát pouze ti studenti, kteří získali zápočet.

Přednášející a učitelé vedoucí počítačová cvičení mají právo oceňovat mimořádné aktivity studentů prémiovými body, které však nejsou nárokové.

Klasifikace úspěšných studentů se řídí studijními předpisy FIT VUT a zásadami ECTS (Evropského kreditového systému). Podmínkou úspěšného absolvování předmětu je pak získání nejméně 50 bodů ze 100 možných.

**Odhad časové náročnosti**



Přibližný odhad časové náročnosti předmětu (6 kreditů) lze stanovit takto:

1 kredit = 25 až 30 hodin práce $\Rightarrow$ 6 kreditů odpovídá 150 až 180 hodinám práce studenta, z toho:

- přednášky 39 hod
- počítačová cvičení 13 hod
- průběžné studium, včetně přípravy příkladů na cvičení cca 60 hod
- příprava na půlsemestrální a závěrečnou zkoušku cca 50 hod

**1.2 Metodické informace**



Předmět Asemblery je povinným předmětem v prvním semestru studia a proto nemůže navazovat na žádný jiný předmět. Získané znalosti však budou výchozím a základním předpokladem pro absolvování celé řady jiných předmětů. Znalost základního jazyka je nezbytným předpokladem dobré práce každého budoucího profesionálního programátora a odborníka v oblasti informačních technologií. Předmět Asemblery prošel na FIT (dříve FEI) více než 30 letým vývojem. Byl nejprve postaven na počítačích ADT a IBM, od roku 1990 je zaměřen na počítače PC s procesory Intel. K tvorbě náplně předmětu sloužily především firemní manuály, protože odborné knihy vycházely pro potřeby předmětu většinou pozdě. Hlavní studijní literatury představovalo po dlouhou dobu několik různých vydání skript "Strojově orientované jazyky", která jsou nyní již prakticky nedostupná. Proto se uvažuje o jejich novém a výrazně přepracovaném přepracovaném vydání, které by respektovalo poslední úpravu obsahu předmětu vyplývající ze

zavedení tříletého bakalářského studijního programu. Původně dominantní orientace na práci v chráněném režimu procesoru se přesunula na práci ve výrazně jednodušším základním režimu procesoru, spojený s výkladem pouze nejdůležitějších instrukcí. Rovněž původně používaný překladač byl nahrazen volně dostupným překladačem a s ním nutně spojeným jazykem. Pro potřeby výuky v počítačových cvičeních byla vytvořena knihovna, umožňující okamžitou praktickou práci.

**Odborná terminologie**



Přesné myšlení je samozřejmým předpokladem každé inženýrské práce a musí se proto opírat o přesné pojmy a správnou terminologií. Odborný žargon, který je často živým komunikačním nástrojem každodenního života, není vhodný ani pro publikační ani pro vyspělé prezentační aktivity. Proto se i text této opory snaží o přesnou a terminologicky správnou prezentaci nových pojmů.

**Anglická terminologie**

V textu se budou často vyskytovat anglické termíny. Jejich první výskyty budou uváděny za příslušnými českými pojmy v závorce a budou psány *kurzívou*.

**Grafická úprava**

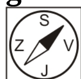
Text je psán s použitím standardního typu Times New Roman, 12 pt. Významné pojmy nebo části textu jsou zvýrazněny **tučně** a/nebo <u>podtržené</u>, v některých případech jsou uvedeny v "uvozovkách". Ty části programových úseků u nichž to bude významné, budou s ohledem na odsazování uvedeny typem `Courier.`
Doplňující poznámky budou psány menším typem Times New Roman, 10 pt.

**Učební cíle a kompetence - specifické**



Studenti se seznámí s architekturou procesorů Intel Pentium (základní režim) včetně jednotky FPU a naučí se používat nejdůležitější celočíselné a FPU instrukce. Dále se naučí jazyk symbolických instrukcí NASM, budou schopni vytvářet programy v tomto jazyku a překládat vytvořené programy do spustitelných programů. Získají i základní vědomosti o předávání řízení a parametrů a o službách operačního systému, a budou schopni tyto získané vědomosti prakticky používat.

**Učební cíle a kompetence - generické**



Studenti získají základní vědomosti o architektuře a činnosti procesoru, které patří k základním znalostem všech odborníků oboru IT. Naučí se řešit jednoduché problémy v jazyku symbolických instrukcí a vést řádnou dokumentaci elementárních počítačových programů.

**Anotace**

Číselné soustavy. Zobrazování celých čísel bez a se znaménkem, aritmetika ve dvojkové soustavě. Strojový jazyk, jazyk symbolických instrukcí, asembler. Architektura procesorů Intel Pentium (registry, organizace hlavní paměti, přerušovací systém). Soubor celočíselných instrukcí. Programování na úrovni strojového jazyka. Jazyk symbolických instrukcí NASM, symbolické instrukce, direktivy, makroinstrukce. Překlad a sestavování. Standardní předávání řízení a parametrů při volání procedur a funkcí. Služby operačního systému. Programování periferií PC (videoRAM, myš, reproduktor). Zobrazování reálných čísel, standard IEEE. Architektura FPU a soubor instrukcí FPU. Programování FPU.

**Přibližný rozpis přednášek**

1. Úvod, číselné soustavy, aritmetika.
2. Strojový jazyk, jazyk symbolických instrukcí, asembler.
3. Základní režim procesorů Pentium: soubor registrů, typy operandů, formát instrukcí, adresování paměti, přerušení.
4. Soubor instrukcí procesorů Pentium. Celočíselné instrukce.
5. Celočíselné instrukce, pokračování.
6. Celočíselné instrukce, pokračování.

7. Zásady programování ve strojovém jazyku, typické řídící konstrukce.
8. Jazyk symbolických instrukcí, asembler NASM.
9. NASM, pokračování.
10. Programové moduly, knihovny, služby operačního systému. Procedury a funkce, standardní předávání řízení a parametrů.
11. FPU procesorů Pentium - architektura.
12. Soubor instrukcí FPU.
13. Příklady programů.

**Studijní literatura**




1. Marek, R.: Assembler pro PC - učíme se programovat v jazyce, Computer Press, 2003, ISBN 80-7226-843-0
2. Carter, P.: Assembly language tutorial, 2002 http://www.drpaulcarter.com/pcasm/

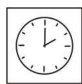

**Poznámka**



Petr Hanáček je autorem kapitol 5 – 8, František Zbořil je autorem kapitol 1 – 4. Autoři budou vděčni za upozornění na nedostatky tohoto textu, které čtenáři mohou poslat na adresy: hanacek@fit.vutbr.cz a zboril@fit.vutbr.cz .

# 2 ČÍSELNÉ SOUSTAVY, ZOBRAZENÍ ČÍSEL A ZNAKŮ



20 hod

Cílem této kapitoly je podat základní informace o číselných soustavách a převodech čísel mezi těmito soustavami, o zobrazení čísel a znaků v pamětech počítačů, především pak o zobrazení bezznaménkových a znaménkových čísel a o základních aritmetických operacích s těmito čísly. Stručně je v této kapitole popsáno i zobrazení BCD čísel a zobrazení čísel v plovoucí řádové čárce.

## 2.1 Číselné soustavy



Libovolné číslo lze zapsat ve tvaru polynomu

$$\text{číslo} = a_{n-1}z^{n-1} + a_{n-2}z^{n-2} + \ldots + a_0 z^0 + a_{-1}z^{-1} + a_{-2}z^{-2} + \ldots \qquad (1)$$

kde $z$ je základ číselné soustavy a $a_i$ jsou číslice soustavy. Základem soustavy může být libovolné přirozené číslo větší než jednička, číslicemi soustavy jsou všechna celá čísla splňující nerovnost $0 \leq a_i < z$ .

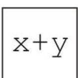
Při zápisu čísla se obvykle používá zkrácený zápis

$$\text{číslo} = \left(a_{n-1}a_{n-2}\ldots a_0.a_{-1}a_{-2}\ldots\right)_z \qquad (2)$$

resp., nemůže-li být pochyby o základu soustavy, pouze

$$\text{číslo} = a_{n-1}a_{n-2}\ldots a_0.a_{-1}a_{-2}\ldots \qquad (3)$$

Celou část čísla od necelé části čísla odděluje řádová tečka.




Příklad: Číslo 101.1 v běžně používané desítkové (dekadické) soustavě vyjadřuje hodnotu sto jedna celých a jednu desetinu. Ve dvojkové (binární) soustavě však stejné číslo vyjadřuje hodnotu pět celých a jednu polovinu, v šestnáctkové (hexadecimální) soustavě dvě stě padesát sedm celých a jednu šestnáctinu, atd.

## 2.2 Převody čísel mezi soustavami

1. Číslo z libovolné soustavy lze převést do desítkové soustavy podle vztahu (1).

   Například převod čísel z předcházejícího příkladu je následující:

   $(101.1)_2 = 1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} = 4 + 1 + 0.5 = 5.5$

   $(101.1)_{16} = 1*16^2 + 0*16^1 + 1*16^0 + 1*16^{-1} = 256 + 1 + 0.0625 = 257.0625$



2. Opačný převod, tj. převod čísla z desítkové soustavy do libovolné jiné soustavy, se provádí odlišným způsobem pro celou část čísla (*cislocele*) a necelou část čísla (*necelecislo*). Nechť *z* je základ soustavy, do které se převod provádí a nechť *m* je požadovaný počet číslic vpravo od řádové tečky (převod necelé části čísla není obecně konečný!). Vlastní převod se pak provádí pomocí následujících dvou algoritmů:

```
n = 0;                              // převod celé části převáděného čísla
dokud (cislocele # 0) opakuj
 { a_n = cislocele mod z;
   cislocele = cislocele div z;
   n = n + 1; }
i = -1;                             // převod necelé části převáděného čísla
dokud (i > -m) opakuj
 { pom = necelecislo * z;
   a_i = trunc(pom);
   necelecislo = pom - a_i ;
   i = i – 1; }
```



Pozn.: operátor **mod** vyhodnocuje zbytek po celočíselném dělení, operátor **div** celočíselný podíl.

Číslo v příslušné soustavě se pak vyjádří pomocí vztahu (2), resp. (3) - má *n* číslic vlevo a *m* číslic vpravo od řádové tečky.

x+y

Příklad: Převod čísla 586.248

a) převod celé části čísla do soustavy

dvojkové:

| 586 / 2 | | |
|---|---|---|
| | **div** | **mod** |
| *i* | *cislocele* | $a_i$ |
| 0 | 293 | 0 |
| 1 | 146 | 1 |
| 2 | 73 | 0 |
| 3 | 36 | 1 |
| 4 | 18 | 0 |
| 5 | 9 | 0 |
| 6 | 4 | 1 |
| 7 | 2 | 0 |
| 8 | 1 | 0 |
| 9 | 0 | 1 |

šestnáctkové:

| 586 / 16 | | |
|---|---|---|
| | **div** | **mod** |
| *i* | *cislocele* | $a_i$ |
| 0 | 36 | 10 = A |
| 1 | 2 | 4 |
| 2 | 0 | 2 |



Číslice 10 až 15 šestnáctkové soustavy se pro jednoznačnost označují písmeny A … F, resp. a … f (10=A=a, 11=B=b, 12=C=c, 13=D=d, 14=E=e, 15=F=f)

b) převod necelé části čísla do soustavy

dvojkové:

| 0.248 ∗ 2 | | |
|---|---|---|
| *i* | $a_i$ | *necelecislo* |
| -1 | 0 | 496 |
| -2 | 0 | 992 |
| -3 | 1 | 984 |
| -4 | 1 | 968 |
| -5 | 1 | 936 |
| -6 | 1 | 872 |
| -7 | 1 | 744 |
| -8 | 1 | 488 |
| -9 | 0 | 976 |
| -10 | 1 | 952 |
| -11 | 1 | 904 |
| -12 | 1 | 808 |
| -13 | … | … |

šestnáctkové:

| 0.248 ∗ 16 | | |
|---|---|---|
| *i* | $a_i$ | *necelecislo* |
| -1 | 3 | 968 |
| -2 | 15=F | 488 |
| -3 | 7 | 808 |
| -4 | … | … |

Tedy: $(586.248)_{10} \approx (1001001010.001111110111)_2 = (24A.3F7)_{16}$



Při převodu z desítkové do jiné soustavy vzniká často chyba, která se zvyšuje s klesajícím počtem převedených číslic vpravo od řádové tečky. Tato chyba se nazývá chybou zkrácení/zanedbání (*Truncation Error*) a je poměrně významná; například pro výše uvedený příklad s uvažováním pouze deseti binárních číslic vpravo od řádové tečky platí

$$(1001001010.0011111101)_2 = (586.24707)_{10}$$

což je číslo, které se od původně převáděného čísla $(586.248)_{10}$ liší téměř o jednu tisícinu!

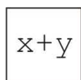
Bezchybně lze převést pouze čísla celá a čísla, pro jejichž necelé části přesně platí

$$\mathit{necelecislo} = a_{-1}z^{-1} + a_{-2}z^{-2} + \ldots + a_{-m}z^{-m}$$

kde *m* je počet převedených číslic v soustavě o základu *z*.



Při převodu čísel mezi dvěma různými číselnými soustavami, pro které platí vztah $z_2 = z_1^j$, kde *j* je přirozené číslo větší než jedna, lze každých *j* číslic čísla v soustavě se základem $z_1$ nahradit jedinou číslicí čísla v soustavě se základem $z_2$ (počítáno v obou směrech od řádové tečky!) a naopak.




Příklad: $16=2^4 \Rightarrow$ $( 0010\ 0100\ 1010 \,.\, 0011\ 1111\ 0111 )_2 = ( 2\ 4\ A \,.\, 3\ F\ 7 )_{16}$

Popsaným způsobem lze převést libovolné desítkové číslo do soustavy s libovolným základem a zpět. Kromě výše popsaných soustav však žádná jiná soustava nemá ve výpočetní technice praktické využití. Pro další výklad je užitečné znát zpaměti následující vzájemné vztahy mezi číslicemi desítkové, dvojkové a šestnáctkové soustavy:

| Desítková soustava | Dvojková soustava | Šestnáctková soustava |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |



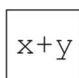
Úkol: Převeďte z desítkové do dvojkové a šestnáctkové soustavy následující čísla: 234.135, 1028.25, 0.1, 350.0 a 15.5.

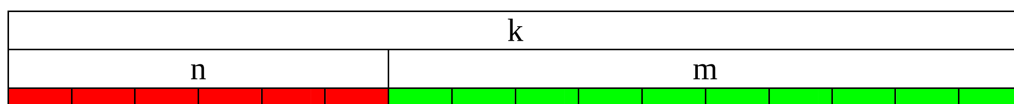
## 2.3 Zobrazení binárních čísel bez znaménka



Při zobrazování čísel se používají pojmy **rozsah, rozlišitelnost a přesnost zobrazení**. Rozsahem zobrazení se rozumí interval ohraničený zleva nejmenším a zprava největším zobrazitelným číslem. Rozlišitelnost zobrazení je dána nejmenším zobrazitelným číslem. Přesnost zobrazení se obvykle udává v počtu platných dekadických číslic, které je možné v daném paměťovém prostoru zobrazit a je nezávislá na hodnotě zobrazovaného čísla.

Zobrazí-li se např. v *k*-bitovém paměťovém prostoru dvojkové číslo bez znaménka (tj. kladné číslo), které má ve shodě s předchozími úvahami *n* míst vlevo a *m* míst vpravo od řádové tečky:




pak rozsah zobrazení je dán intervalem ⟨0, ($2^n$ – $2^{-m}$)⟩, rozlišitelnost hodnotou $2^{-m}$ a přesnost zobrazení přibližně vztahem $p \approx k * \log(2)$, odvozeným z předpokladu, že číslo ≈ $10^p \approx 2^k$, kde *p* je počet dekadických cifer nutný pro zobrazení převáděného čísla.

Je zřejmé, že posunutím řádové tečky o jedno místo vlevo, tj. vydělením původního čísla dvěma, se sníží dvakrát rozsah zobrazení a zvýší se dvakrát rozlišitelnost zobrazení. Naopak posunutím řádové tečky o jedno místo vpravo, tj. vynásobením původního čísla dvěma, se zvýší dvakrát rozsah zobrazení a sníží se dvakrát rozlišitelnost zobrazení. Přesnost zobrazení se přitom přirozeně nemění.

Hodnoty *m*, resp. *N* mohou být nulové i záporné, vždy však musí platit $k = m + n$. Tím je možné zobrazovat i příliš velká, nebo malá čísla. V praxi se ve výše uvedeném zobrazení, označovaném jako zobrazení s pevnou řádovou tečkou) zobrazují většinou pouze celá čísla ($m = 0$, $k = n$), pro která je rozsah zobrazení dán intervalem ⟨0, ($2^n$ – 1)⟩ a rozlišitelnost zobrazení hodnotou 1. Pro reálná čísla se pak používá zobrazení nazývané zobrazením v plovoucí řádové tečce, které bude podrobně popsáno v kapitole 2.6.

Číslice binární soustavy se označuje anglickým termínem *bit* (***bi**nary **d**igit*) a mají význam nejmenší jednotky informace. Zkratkou pro bit je malé písmeno *b*. Pro větší uskupení bitů se pak používají následující názvy:

| Počet bitů | Rozsah zobrazení | Přesnost zobrazení | Označení |
|---|---|---|---|
| 4 | <0, 15> | 1.2 | Nibble |
| 8 | <0, 255> | 2.4 | Byte (slabika) |
| 16 | <0, 65535> | 4.8 | Word (slovo) |
| 32 | <0, 4294967295> | 9.6 | Doubleword (dvouslovo) |
| 64 | <0, ≈ $1.84 \cdot 10^{19}$> | 19.2 | Quadword (čtyřslovo) |
| 128 | <0, ≈ $3.40 \cdot 10^{38}$> | 38.5 | Double Quadword (dvojice čtyřslov) |

**Aritmetika pro binární čísla bez znaménka**

Základní aritmetické operace ve dvojkové soustavě jsou podobné základním aritmetickým operacím v desítkové soustavě.

Při aritmetických operacích prováděných v počítači se však výsledek musí vejít do předem určeného paměťového prostoru, tzn., že výsledek operace musí být v tomto prostoru zobrazitelný. V opačném případě obdržíme nesprávný výsledek a musíme být proto schopni takové situace jednoznačně identifikovat.



**Sečítání:**

0 + 0 = 0
0 + 1 = 1 + 0 = 1
1 + 1 = 10 ( $10_2 = 2_{10}$ , tj. 0 a přenos do vyššího řádu !)
1 + 1 + 1 = 11 ( $11_2 = 3_{10}$ , tj. 1 a přenos do vyššího řádu !)
1 + 1 + 1 + 1 = 100
atd.

x+y

Příklady sečítání dvou binárních čísel v osmibitovém prostoru ($m$=0, $k$=$n$=8):

Zobrazitelný součet:

```
  00101100     desítkově:   44
+ 01011010                 +90
  10000110                 134
```

Nezobrazitelný součet:

```
  10110100              desítkově:   180
+ 10110100                          +180
 1|01101000  = 104₁₀      #          360
```

Při nezobrazitelném součtu došlo k přenosu z nejvyššího zobrazitelného bitu (má hodnotu 256), a zobrazený výsledek je pak o tuto hodnotu menší (360–256=104).

!

Nesprávný výsledek sečítání dvou bezznaménkových binárních čísel je indikován přenosem z nejvyššího bitu, který se nazývá *CF* (*Carry Flag*)!

!

**Odečítání:** 0 – 0 = 0
1 – 0 = 1
1 – 1 = 0
0 – 1 = 1 (10 – 1, tj. 1 a výpůjčka z vyššího řádu !)

x+y

Příklady odečítání dvou binárních čísel v osmibitovém prostoru ($m$=0, $k$=$n$=8):

Zobrazitelný rozdíl:

```
  01011010     desítkově:  90
– 00101100                – 44
  00101110                  46
```

Nezobrazitelný rozdíl:

```
 1|00101100              desítkově:  44
– 01011010                          – 90
  11010010 = 210₁₀         #        – 46
```

Při nezobrazitelném rozdílu došlo k výpůjčce („vypůjčený bit“ má hodnotu 256 a tak zobrazený výsledek je o tuto hodnotu větší: 256+(–46)=210).

!

Nesprávný výsledek odečítání dvou bezznaménkových binárních čísel je indikován výpůjčkou - přenosem do nejvyššího bitu, který se opět nazývá *CF*!

!

**Násobení:** 0 ∗ 0 = 0
1 ∗ 0 = 0 ∗ 1 = 0
1 ∗ 1 = 1

x+y

Příklady násobení dvou binárních čísel v osmibitovém prostoru ($m$=0, $k$=$n$=8):

Zobrazitelný součin (11 ∗ 14 = 154):

```
00001011 * 00001110
00000000
 00000000
  00000000
   00000000
    00001011
     00001011
      00001011
       00000000
```

```
                                              0000000010011010   = 154
```

Nezobrazitelný součin (18 ∗ 17 = 306):

```
00010010 * 00010001
00000000
 00000000
  00000000
   00010010
    00000000
     00000000
      00000000
       00010010
0000000100110010    = 50 # 306 (256 + 50 = 306)
```

Při násobení dvou binárních čísel nelze zobrazitelnost, tj. správnost výsledku zjistit podobně jednoduchým způsobem jako u operací sečítání a odečítání. Navíc se násobením obecně mění i pozice řádové tečky. Proto se prakticky vždy ukládá součin do dvojnásobného paměťového prostoru, než je použit pro zobrazení každého z činitelů a v tomto prostoru je pak součin zobrazitelný vždy (je zřejmé, že tento součin může být zobrazen v původním paměťovém prostoru a s původně umístěnou řádovou tečkou, jestliže obsahuje alespoň ($k$ – $m$) vedoucích nul).

**Dělení:** 0 : 0 = nepovolená operace (dělení nulou)
1 : 0 = nepovolená operace (dělení nulou)
0 : 1 = 0
1 : 1 = 1



Dělení (přesněji celočíselné dělení, kdy podílem je celé číslo!) je opačnou aritmetickou operací k operaci násobení a dělenec se proto obvykle ukládá do dvojnásobného paměťového prostoru, než je použit pro zobrazení dělitele, podílu a zbytku (tzv. modula).

Příklady dělení dvou binárních čísel v osmibitovém prostoru ($m$=0, $k$=$n$=8):

Zobrazitelný podíl dvou binárních čísel (například 100 : 11 = 9 (celočíselný podíl) a 1 (zbytek/modulo)):

x+y

```
 0000000001100100 : 00001011 = 000001001
–00001011
 –00001011
  –00001011
   –00001011
    –00001011
     –00001011
      00000001100
      –00001011
       –00001011
        –00001011
         00000001   ---  (zbytek/modulo)
```

Nezobrazitelný podíl dvou binárních čísel (např. 300 : 1 = 300 (nezobrazitelný celočíselný podíl) a 0 (zbytek/modulo)):

```
 0000000100101100 : 00000001 = 1xxxxxxxx
–00000001
 00000000
```

U celočíselného dělení musí být podíl zobrazitelný v *k* bitovém prostoru, tzn., že musí mít hodnotu v rozsahu $\langle 0, (2^k - 1)\rangle$, zbytek po dělení (modulo) je samozřejmě zobrazitelný vždy. Pokud má podíl větší hodnotu než $(2^k - 1)$, je tento podíl nezobrazitelný a výsledek dělení je pak nesprávný.



Úkoly:

Převeďte na osmibitová binární čísla bez znaménka následující dvojice dekadických čísel (17, 23), (118, 150), (200, 12), (3, 7) a tyto pak sečtěte, odečtěte, vynásobte a podělte. Určete, které výsledky jsou a které nejsou zobrazitelné v daném osmibitovém prostoru.

## 2.4 Zobrazení binárních čísel se znaménkem

Čísla se znaménkem lze zobrazovat pouze prostřednictvím transformací, kdy zobrazovaná čísla bez znaménka reprezentují v nějakém intervalu čísla záporná. Označí-li se symbolem *x* zobrazované číslo se znaménkem a symbolem *X* jeho kladný obraz (číslo bez znaménka), pak zobrazování čísel se znaménkem se provádí nejčastěji některou z následujících tří transformací:



a) $X = x$ pro *x* z intervalu $\langle 0, (2^{(n-1)} - 2^{-m}) \rangle$
$X = 2^{(n-1)} - x$ pro *x* z intervalu $\langle -(2^{(n-1)} - 2^{-m}), 0 \rangle$

b) $X = x$ pro *x* z intervalu $\langle 0, (2^{(n-1)} - 2^{-m}) \rangle$
$X = 2^{n} + x$ pro *x* z intervalu $\langle -2^{(n-1)}, -2^{-m} \rangle$

c) $X = 2^{(n-1)} + x$ pro *x* z intervalu $\langle -2^{(n-1)}, (2^{(n-1)} - 2^{-m}) \rangle$

Zobrazení s uvedenými transformacemi se často nazývají kódy:

a) kód přímý
c) kód doplňkový
d) kód transformované nuly

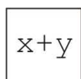
### Převody čísel

Převody čísel se znaménkem na bezznamémková zobrazovaná čísla jsou poměrně snadné a budou ukázány na příkladech pro $k = n = 8$ bitů, $m = 0$. Pak:

### Kód přímý

1. Kód přímý (z definice):

X = x pro *x* z intervalu <0 , 127>
X = 128 – x pro *x* z intervalu <–127, 0>




Příklad převodu (např. čísel 28, –28 a 0):

| | | | |
|---|---|---|---|
| x = 28 | → | X = x = 28 | 00011100 |
| x = –28 | → | X = 128 – (–28) = 156 | 10011100 |
| x = 0 | → | X = x = 0 | 00000000 |
| x = 0 | → | X = 128 – 0 = 128 | 10000000 |

Číslo kladné se od stejného čísla záporného v přímém kódu liší pouze hodnotou nejvyššího bitu (0 pro kladná čísla, 1 pro záporná čísla). Nula má dva rovnocenné

obrazy.

**Kód doplňkový**

2. Kód doplňkový (z definice):

X = x pro *x* z intervalu <0 , 127>
X = 256 + x pro *x* z intervalu <–128, –1>

x+y

Příklad převodu (opět čísel 28, –28 a 0):

| | | | |
|---|---|---|---|
| x = 28 | → | X = x = 28 | 00011100 |
| x = -28 | → | X = 256 + (-28) = 228 | 11100100 |
| x = 0 | → | X = x = 0 | 00000000 |

Číslo kladné se od stejného čísla záporného v doplňkovém kódu liší hodnotou nejvyššího bitu (0 pro kladná čísla a nulu, 1 pro záporná čísla) i hodnotou všech ostatních bitů. Prakticky se číslo **s opačným znaménkem** (tj. z kladného záporné, nebo ze záporného kladné !!!) získá inverzí hodnot všech bitů a aritmetickým přičtením jedničky k nejnižšímu bitu.

| | |
|---|---|
| 00011100 | $(28)_{10}$ |
| 11100011 | inverze |
| + 1 | |
| 11100100 | $(-28)_{10}$ |
| 00011011 | inverze |
| + 1 | |
| 00011100 | $(28)_{10}$ |

**Kód transformované nuly**

3. Kód transformované nuly (z definice):

X = 128 + x pro *x* z intervalu <–128, 127>

x+y

Příklad převodu (opět čísel 28, –28 a 0):

| | | | |
|---|---|---|---|
| x = 28 | → | X = 128 + 28 = 156 | 10011100 |
| x = –28 | → | X = 128 + (–28) = 100 | 01100100 |
| x = 0 | → | X = 128 + 0 = 128 | 10000000 |

Číslo kladné se od stejného čísla záporného v kódu transformované nuly liší opět hodnotou nejvyššího bitu (tentokrát 1 pro kladná čísla, 0 pro záporná čísla) i hodnotou ostatních bitů. Prakticky se číslo **s opačným znaménkem** získá z čísla v doplňkovém kódu změnou hodnoty nejvyššího bitu (platí pouze pro výše uvedenou definici, neboť tento kód se používá i v modifikovaných definicích!).

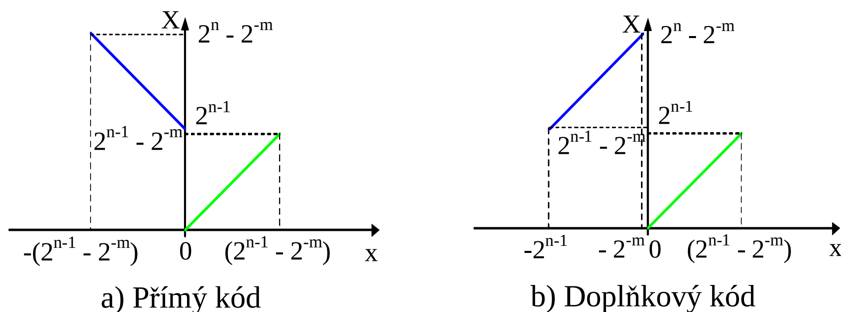
Grafická znázornění výše uvedených transformací/kódů jsou uvedena na obr. 2.1.

Informaci o znaménku čísla ve všech uvedených kódech nese nejvyšší bit, který se proto nazývá bitem znaménkovým (*Sign bit*) nebo také bitem nejvíce významným (*Most Significant Bit* - MSB).

Všechny ostatní bity, které nesou informaci o hodnotě čísla se pak souhrnně označují jako bity významové. Bit nejvíce vpravo má nejmenší význam (*Least Significant Bit* – LSB).

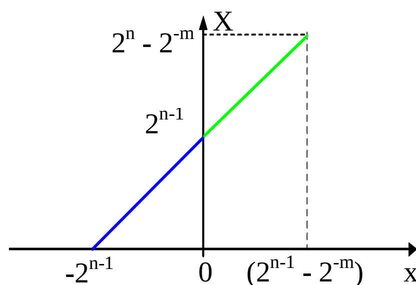
Přímý kód a kód transformované nuly se obvykle používají pro zobrazování reálných čísel v pohyblivé řádové tečce (kap. 2.6). Pro zobrazování celých čísel se prakticky výhradně používá pouze kód doplňkový.





a) Přímý kód

b) Doplňkový kód




c) Kód transformované nuly

Obr. 2.1 Grafická znázornění transformací čísel se znaménkem na čísla bez znaménka

**Aritmetika v doplňkovém kódu pro binární čísla se znaménkem**

Ve výpočetní technice má zásadní důležitost kód doplňkový, především pro jednoduchost svých aritmetických operací:

Nechť $k = n = 4$, $m = 0$ $\Rightarrow$ $x \in \langle -8, 7 \rangle$ a nechť *CF* značí přenos/výpůjčku z/do nejvyššího bitu a *P* přenos/výpůjčku do/z nejvyššího bitu:

x+y

**Sečítání** (může nastat 6 různých případů):

1) Dvě kladná čísla, výsledek je zobrazitelný

| | | | |
|---|---|---|---|
| 3 | | 0011 | |
| + 2 | | 0010 | |
| 5 | | 0101 = $5_{10}$ | CF = 0, P = 0 |

2) Dvě kladná čísla, výsledek není zobrazitelný

| | | | |
|---|---|---|---|
| 5 | | 0101 | |
| + 6 | | 0110 | |
| 11 | # | 1011 = $-5_{10}$ | CF = 0, P = 1 |

3) Dvě záporná čísla, výsledek je zobrazitelný

| | | | |
|---|---|---|---|
| –4 | | 1100 | |
| + –2 | | 1110 | |
| –6 | | 11010 = $-6_{10}$ | CF = 1, P = 1 |

4) Dvě záporná čísla, výsledek není zobrazitelný

| | | | |
|---|---|---|---|
| –4 | | 1100 | |
| + –6 | | 1010 | |
| –10 | # | 10110 = $6_{10}$ | CF = 1, P = 0 |

5) Kladné a záporné číslo, výsledek je kladný (musí být zobrazitelný)

| | | | |
|---|---|---|---|
| 5 | | 0101 | |
| + –2 | | 1110 | |
| 3 | | 10011 = $3_{10}$ | CF = 1, P = 1 |

6) Kladné a záporné číslo, výsledek je záporný (musí být zobrazitelný)

| | | | |
|---|---|---|---|
| 4 | | 0100 | |
| + –6 | | 1010 | |
| –2 | | 1110 = $–2_{10}$ | CF = 0, P = 0 |

**Odečítání** (může nastat 8 různých případů):

1) Dvě kladná čísla, výsledek je kladný (musí být zobrazitelný)

| | | | |
|---|---|---|---|
| 7 | | 0111 | |
| – 2 | | 0010 | |
| 5 | | 0101 = $5_{10}$ | CF =0, P = 0 |

2) Dvě kladná čísla, výsledek je záporný (musí být zobrazitelný)

| | | | |
|---|---|---|---|
| 5 | | 10101 | |
| – 6 | | 0110 | |
| –1 | | 1111 = $–1_{10}$ | CF = 1, P = 1 |

3) Dvě záporná čísla, výsledek je záporný (musí být zobrazitelný)

| | | | |
|---|---|---|---|
| –4 | | 11100 | |
| – –2 | | 1110 | |
| –2 | | 1110 = $–2_{10}$ | CF = 1, P = 1 |

4) Dvě záporná čísla, výsledek je kladný (musí být zobrazitelný)

| | | | |
|---|---|---|---|
| –4 | | 1100 | |
| – –6 | | 1010 | |
| 2 | | 0010 = $2_{10}$ | CF = 0, P = 0 |

5) Kladné a záporné číslo, výsledek je kladný a zobrazitelný

| | | | |
|---|---|---|---|
| 5 | | 10101 | |
| – –2 | | 1110 | |
| 7 | | 0111 = $7_{10}$ | CF = 1, P = 1 |

6) Kladné a záporné číslo, výsledek je záporný a zobrazitelný

| | | | |
|---|---|---|---|
| –6 | | 11010 | |
| – –4 | | 1100 | |
| –2 | | 1110 = $–2_{10}$ | CF = 1, P = 1 |

7) Kladné a záporné číslo, výsledek je záporný a není zobrazitelný

| | | | |
|---|---|---|---|
| –6 | | 1010 | |
| – 4 | | 0100 | |
| –10 | # | 0110 = $6_{10}$ | CF = 0, P = 1 |

8) Kladné a záporné číslo, výsledek je kladný a není zobrazitelný

| | | | |
|---|---|---|---|
| 5 | | 10101 | |
| – –7 | | 1001 | |
| 12 | # | 0100 = $4_{10}$ | CF = 1, P = 0 |



Ve všech uvedených příkladech byl výsledek správný (zobrazitelný), pokud byly oba přenosy *CF* i *P* stejné a výsledek byl nesprávný (nezobrazitelný), pokud došlo pouze k jednomu z těchto dvou přenosů. Zjištěná vlastnost doplňkového kódu (lze

ji snadno matematicky dokázat) je při operacích sečítání a odečítání využívána k nastavení příznaku přetečení *OF* (*Overflow Flag*) :

$$(P \# CF) \rightarrow OF$$

**Násobení:** pro jednoduchost můžeme předpokládat, že se násobení provádí s kladnými čísly a znaménko výsledku (součinu) se změní, pokud byl jeden a právě jeden z činitelů záporný.

**Dělení (celočíselné):** pro jednoduchost můžeme opět předpokládat, že se dělení provádí s kladnými čísly a že znaménko výsledku (podílu) se změní, pokud byl dělitel nebo dělenec (tj. pouze jeden z nich!) záporný. Znaménko zbytku je dáno znaménkem dělence.

U celočíselného dělení se definují dvě operace: *div* (celočíselný podíl) a *mod* (zbytek po celočíselném dělení):

| a | b | a *div* b | a *mod* b |
|---|---|---|---|
| 114 | 5 | 22 | 4 |
| -114 | 5 | -22 | -4 |
| 114 | -5 | -22 | 4 |
| -114 | -5 | 22 | -4 |



Úkoly:

Převeďte na osmibitová binární čísla se znaménkem následující dvojice dekadických čísel (-17, 23), (118, -15), (-20, -12), (3, 7) a tyto pak sečtěte, odečtěte, vynásobte a podělte. Určete, které výsledky jsou a které nejsou zobrazitelné v daném osmibitovém prostoru.

## 2.5 Zobrazení binárně kódovaných dekadických čísel (BCD)

Zvláštním a málo používaným způsobem zobrazování čísel v počítači je tzv. BCD kód (Binary Coded Decimal), ve kterém je každá desítková číslice zobrazena samostatně na čtyřech bitech ve dvojkovém kódu. Případné znaménko je pak obvykle uloženo v prvním Byte čísla (0 pro kladné číslo, 1 pro záporné číslo)

Příklad: $(586.248)_{10} = (0101\ 1000\ 0110\ .\ 0010\ 0100\ 1000)_{BCD}$

## 2.6 Zobrazení reálných čísel v pohyblivé (plovoucí) řádové tečce

Zobrazení reálných čísel v pohyblivé (plovoucí) řádové tečce spočívá v rozložení čísla na mantisu (*Fraction*) a exponent a v samostatném zobrazení/uložení těchto dvou částí čísla. Zobrazované číslo je pak dáno vztahem

$$\text{číslo} = \text{mantisa} * \text{základ}^{\text{exponent}}.$$

Základ je explicitní a v dále popsaném standardu IEEE je roven základu dvojkové soustavy, tj. číslu 2. Ve zmíněném standardu je mantisa zobrazena v přímém kódu a exponent v (modifikovaném) kódu transformované nuly.

I když s čísly uloženými v tomto formátu budeme pracovat až v kapitole 8 „FPU procesorů Pentium, princip práce s reálnými čísly“, z důvodu systematičnosti této studijní opory uvedeme tři používané varianty standardu IEEE na tomto místě:

**Single precision**



**Single–precision** (jednoduchá přesnost, 32 bitů):

| 31 | 30 ... 23 | 22 ... 0 |
|---|---|---|
| S | Exponent e | $f_{22}f_{21}$ … Mantisa f … $f_0$ |

| | Exponent | Mantisa | Číslo |
|---|---|---|---|
| 1 | 0 < e < 255 | f = libovolná | $(-1)^S\ 2^{(e-127)}1.f_{22}f_{21}\ldots f_0$ |
| 2 | e = 0 | f = 0 | 0 |
| 3 | e = 0 | f # 0 | $(-1)^S\ 2^{(-126)}0.f_{22}f_{21}\ldots f_0$ |
| 4 | e = 255 | f = 0 | ∞ |
| 5 | e = 255 | f # 0 | NaN (Not a Number) |

Na řádku 1 je uvedeno tzv. normalizované zobrazení uloženého čísla (též normalizované číslo) – jde o zobrazení, jehož mantisa obsahuje vlevo od řádové tečky jedničku (která je implicitní a proto se neukládá) a číslo je proto uloženo s maximální možnou přesností. Na řádku 2 je uveden tvar nulového čísla. Řádek 3 ukazuje nenormalizované zobrazení (též nenormalizované číslo - vysvětleno níže). Na řádku 4 je uvedeno zobrazení čísla, které je považováno za „nekonečně velké“ a konečně na řádku 5 je uvedeno zobrazení, které říká, že uložený objekt není číslem (například výsledek odmocniny ze záporného čísla).

Pro vysvětlení pojmů normalizované a nenormalizované zobrazení (číslo) nechť slouží následující příklad zobrazení čísla $5.5_{10} = 101.1_2$. Pak zobrazení (číslo)

$$f = 1.011,\ e = 129,\ \text{tj. číslo} = 1.375 * 2^{(129-127)} = 1.375 * 4 = 5.5$$

je normalizované avšak zobrazení

$$f = 0.1011,\ e = 130,\ \text{tj. číslo} = 0.6875 * 2^{(130-127)} = 0.6875 * 8 = 5.5$$
$$f = 0.01011,\ e = 131,\ \text{tj. číslo} = 0.34375 * 2^{(131-127)} = 0.34375 * 16 = 5.5$$

jsou nenormalizovaná - posunem mantisy a dekrementací exponentu je možné tato zobrazení normalizovat.

Denormalizované číslo je pak tak malé číslo, že již normalizovat nelze. Obecně tím dochází ke ztrátě přesnosti.

**Double precision**



**Double–precision** (dvojnásobná přesnost, 64 bitů)**:**

| 63 | 62 ... 52 | 51 ... 0 |
|---|---|---|
| S | Exponent e | $f_{51}f_{50}$ … Mantisa f … $f_0$ |

| | Exponent | Mantisa | Číslo |
|---|---|---|---|
| 1 | 0 < e < 2047 | f = arbitrary | $(-1)^S\ 2^{(e-1023)}1.f_{51}f_{50}\ldots f_0$ |
| 2 | e = 0 | f = 0 | 0 |
| 3 | e = 0 | f # 0 | $(-1)^S\ 2^{(-1022)}0.f_{51}f_{50}\ldots f_0$ |
| 4 | e = 2047 | f = 0 | ∞ |
| 5 | e = 2047 | f # 0 | NaN (Not a Number) |

Zobrazení na jednotlivých řádcích Double-precision odpovídají zobrazením uvedeným na stejných řádcích u zobrazení Single-precision.

Poznamenejme, že rozsah zobrazení je přibližně $\langle -2^{127}, 2^{127}\rangle$, resp. $\langle -2^{1023}, 2^{1023}\rangle$, rozlišitelnost normalizovaných čísel $2^{-127}$, resp. $2^{-1023}$, rozlišitelnost nenormalizovaných čísel $2^{-150}$, resp. $2^{-1075}$ a přesnost 7, resp. 16 dekadických cifer pro zobrazení Single-precision, resp. pro zobrazení Double-precision.

## 2.7 Zobrazení znaků

Znaky lze zobrazovat transformací na bezznaménková čísla. Číslo reprezentující znak se nazývá ordinálním číslem tohoto znaku. Různé transformace/kódy přiřazují jednotlivým znakům různá ordinální čísla, všechny by však měly splňovat následující požadavky:

- Nejnižší 4 bity ordinálního čísla číslice musí reprezentovat binární hodnotu této číslice.
- Písmeno větší v abecedě musí být reprezentováno větším ordinálním číslem.



Znaky se donedávna zobrazovaly výhradně na jedné slabice (8 bitů), v současné době se začíná stále více prosazovat jejich zobrazování na dvou slabikách (16 bitů). V předmětu IAS budeme používat nejpoužívanější klasické zobrazení, tzv. kód ASCII (American Standard Code for Information Interchange):

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 … F |
|---|---|---|---|---|---|---|---|---|---|
| 0 | NUL | DLE | SP | 0 | @ | P | ` | p | |
| 1 | SOH | DC1 | ! | 1 | A | Q | a | q | |
| 2 | STX | DC2 | " | 2 | B | R | b | r | |
| 3 | ETX | DC3 | # | 3 | C | S | c | s | |
| 4 | EOT | DC4 | $ | 4 | D | T | d | t | |
| 5 | ENQ | NAK | % | 5 | E | U | e | u | |
| 6 | ACK | SYN | & | 6 | F | V | f | v | Různé |
| 7 | BEL | ETB | ' | 7 | G | W | g | w | národní |
| 8 | BS | CAN | ( | 8 | H | X | h | x | znaky |
| 9 | HT | EM | ) | 9 | I | Y | i | y | |
| A | LF | SUB | * | : | J | Z | j | z | |
| B | VT | ESC | + | ; | K | [ | k | { | |
| C | FF | FS | , | < | L | \ | l | \| | |
| D | CR | GS | - | = | M | ] | m | } | |
| E | SO | RS | . | > | N | ^ | n | ~ | |
| F | SI | US | / | ? | O | _ | o | DEL | |

Hexadecimální číslice nad tabulkou určuje první čtveřici bitů (*Nibble*), hexadecimální číslice ve sloupci před tabulkou určují druhou čtveřici ordinálních čísel příslušných znaků. Tak např. ordinální číslo $41_{16} = 01000001_2$ reprezentuje znak "A", ordinální číslo $37_{16} = 00110111_2$ reprezentuje znak "7", atd.

Znaky v prvních dvou sloupcích jsou znaky řídicími. Dále se setkáme pouze se znaky *BEL* (*Bell* – zvonek), *ESC* (*Escape* – únik, zrušení), *LF* (*Line Feed* – plný řádek, přechod na nový řádek), *CR* (*Carriage Return*, návrat (vozíku na starých psacích strojích) na počátek řádku). Znak *SP* (*Space*) znamená mezeru.

## 2.8 Shrnutí



Z informací uvedených v této kapitole jsou pro úspěšné absolvování předmětu nejdůležitější: Praktická znalost vzájemných převodů čísel mezi desítkovou, dvojkovou a hexadecimální číselnou soustavou, znalost pojmů *rozsah*, *rozlišitelnost* a *přesnost* zobrazení, znalost aritmetických operací sčítání a odečítání bezznaménkových i znaménkových (v doplňkovém kódu) čísel a znalost principů zobrazení znaků a zobrazení čísel v plovoucí řádové čárce.



# 3 PRINCIP ČINNOSTI POČÍTAČE, STROJOVÝ JAZYK, SYMBOLICKÝ JAZYK, ASEMBLER



20 hod

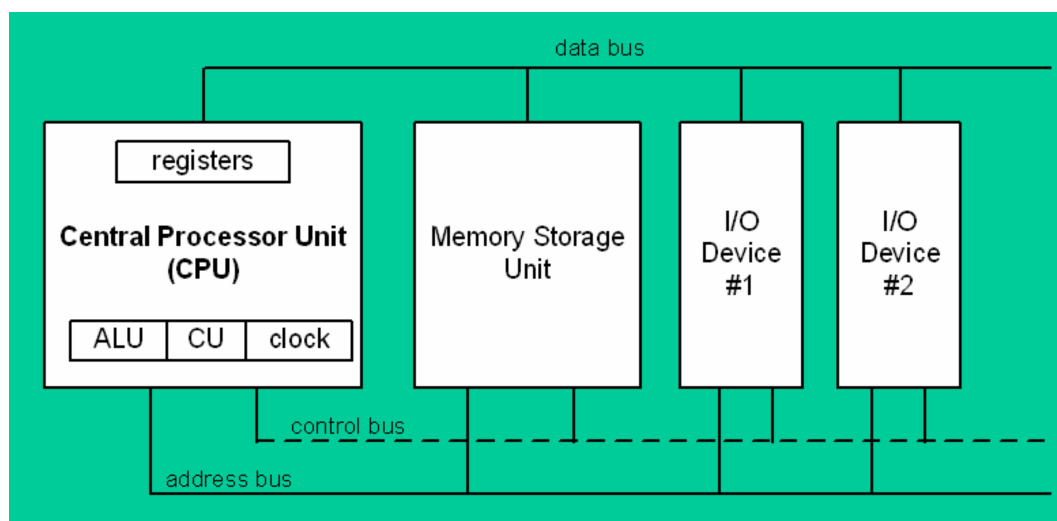
Cílem této kapitoly je poskytnout základní informace o činnosti počítače a na příkladech dvou hypotetických počítačů vysvětlit principy strojového jazyka (strojových instrukcí), symbolického jazyka (symbolických instrukcí a direktiv) a principy práce asembleru – překladače programů zapsaných v symbolickém jazyku do posloupností strojových instrukcí a dat.

## 3.1 Princip činnosti počítače



Zjednodušené blokové schéma počítače je uvedeno na Obr. 3.1. K jeho základním částem, podsystémům, přes různé odlišnosti v konkrétních aplikacích, patří:

- aritmeticko-logická jednotka (*Arithmetic and Logic Unit, ALU*)
- řadič / řídící jednotka (*Control Unit, CU*)
- zdroj taktovacích impulsů/hodin (*clock*)
- registry (*registers*)
- hlavní paměť (*Memory Storage Unit / Main Memory*)
- vstupní a výstupní zařízení (*Input/Output Devices, I/O Devices*)




Obr. 3.1 Zjednodušené blokové schéma počítače

První čtyři uvedené podsystémy tvoří základní jednotku (*Central Processing Unit, CPU*), které se zkráceně říká procesor (*Processor*).

Všechny podsystémy procesoru a počítače spolu komunikují prostřednictvím adresové, datové a řídící sběrnice (*Address bus, Data bus and Control bus*).

K významným registrům CPU patří střádač (*Accumulator, A*), registr ukazatele instrukcí (*Instruction pointer register, IPR*) a instrukční registr (*Instruction register, IR*).



Nejmenší adresovatelnou jednotkou hlavní paměti je obvykle jedna slabika (Byte). Kapacita hlavní i vnějších pamětí se udává v počtu slabik s příponami (dnes prakticky pouze do Tera):

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| k | kilo | = | $2^{10}$ | slabik | P | Peta | = | $2^{50}$ | slabik |
| M | Mega | = | $2^{20}$ | slabik | E | Exa | = | $2^{60}$ | slabik |
| G | Giga | = | $2^{30}$ | slabik | Z | Zetta | = | $2^{70}$ | slabik |
| T | Tera | = | $2^{40}$ | slabik | Y | Yotta | = | $2^{80}$ | slabik |

Činnost počítače je proces postupné transformace vstupních údajů na údaje výstupní, který je řízený posloupností příkazů předem vytvořeného a do hlavní paměti počítače vloženého programu. Jednotlivé příkazy tohoto programu se nazývají instrukce a určují jaké operace a s jakými operandy se budou provádět. Každá instrukce obsahuje operační kód, který jednoznačně určuje operaci a délku instrukce, dále může obsahovat operandy, nebo jejich adresy.

Úplný soubor všech instrukcí daného procesoru se nazývá strojový kód/jazyk a je pro různé počítače značně rozdílný.

V hlavní paměti jsou uloženy dva typy objektů – data, se kterými se pracuje a instrukce, které tuto práci (výpočet) řídí a oba tyto typy objektů představují libovolné (tj. i stejné) kombinace různě velkých binárních čísel. Procesor pak za instrukci považuje vždy ten a jen ten objekt, na který ukazuje registr ukazatele instrukcí!

Činnost počítače při výpočtu, kterou řídí řadič *CU*, je následující:



1. Do instrukčního registru *IR* se uloží instrukce - obsah paměťového místa, které je adresováno registrem ukazatele instrukcí *IPR* (délka instrukce je určena operačním kódem této instrukce).
2. Nastaví se nový obsah registru ukazatele instrukcí *IPR* (ukazuje na následující instrukci programu).
3. Obsah instrukčního registru *IR* je dekódován, t.j. určí se požadovaná operace a adresy příslušných operandů. Operandy mohou být ve střádači *A*, ostatních registrech *CPU*, nebo v hlavní paměti.
4. Provede se určená operace v *ALU* (u instrukcí skokových se nastaví nový obsah registru ukazatele instrukcí *IPR*). Výsledek operace se obvykle ukládá do střádače *A*, nebo do hlavní paměti.
5. Pokud nebyla provedená instrukce pokynem k zastavení procesoru, pokračuje se znovu od bodu 1.

## 3.2 Strojový jazyk, symbolický jazyk, asembler

Pro další výklad použijeme dva hypotetické počítače.

**Hypotetický počítač č. 1**
Obsahuje hlavní paměť s kapacitou 32 slabik a procesor, který má jediný registr-střádač, a který pracuje s jednoslabikovými instrukcemi následujícího formátu:



| Č.bitu | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | operační kód | | | adresa operandu | | | | |

Jednotlivé instrukce provádí následující operace:

| Operační Kód | Symbolický zápis | Popis operace |
|---|---|---|
| 000 | NOP | prázdná operace (no operation) |
| 001 | LOAD adr | ulož operand z paměti (adresy *adr*) do střádače |
| 010 | ADD adr | přičti operand z paměti (adresy *adr*) ke operandu ve střádači |
| 011 | SAVE adr | ulož obsah střádače do paměti (na adresu *adr*) |
| 100 | NEG | změň znaménko čísla ve střádači |
| 101 | JMP adr | skoč na adresu *adr* |
| 110 | JN adr | skoč na adresu *adr*, je-li číslo ve střádači záporné |
| 111 | HALT | stop |

Programování přímo ve strojovém kódu je prakticky nemožné - například program, jehož první instrukce má být uložena v paměti na adrese 5, který má provádět součet dvou bezznaménkových osmibitových čísel uložených v paměti na adresách 20 a 21, a který má uložit výsledek na adresu 25 (bez kontroly jeho správnosti), by vypadal takto:

| | |
|---|---|
| 00101 | 00110100 |
| 00110 | 01010101 |
| 00111 | 01111001 |
| 01000 | 11100000 |

Nepochybně čitelnější je zápis předcházejícího programu v symbolickém jazyku

| | |
|---|---|
| 00101 | LOAD 20 |
| 00110 | ADD 21 |
| 00111 | SAVE 25 |
| 01000 | HALT |



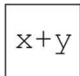
Překlad programu zapsaného v symbolickém jazyku (*Assembly language*) do strojového jazyka provádí překladač (*Assembler*). V češtině se oba předchozí pojmy často nerozlišují a označují se stejným slovem „asembler".



Aby překladač byl překlad schopen provést, musí mu být poskytnuty některé dodatečné informace, které se označují jako direktivy (pokyny pro překladač).

Pro náš hypotetický počítač a následující jednoduché programy stačí definovat pouze čtyři direktivy:

| | |
|---|---|
| ORG adr | definice počáteční adresy programu |
| END adr | označení konce programu a definice startovací adresy |
| DEC number | definice dekadické konstanty |
| RESB n | reservace n slabik paměti |




**Příklad 1:** Program pro příkaz: **if (A >= B) C = A; else C = B;** (tj. pro příkaz **C =** ***max*****(A, B);**), čísla A a B jsou čísla se znaménky:

```
          ORG 5          ; začátek programu v paměti
START:    LOAD A         ; A do střádače
          JN ANEG        ; je-li záporné, tak skok na ANEG
          LOAD B         ; A je kladné, tak B do střádače
          JN THEN        ; je-li B záporné, tak A je větší
          JMP TEST       ; obě jsou kladná, nutný test
ANEG:     LOAD B         ; A je záporné, B do střádače
          JN TEST        ; obě jsou záporná, nutný test
          JMP ELSE       ; B je kladné a proto je větší
TEST:     NEG            ; ve střádači je B ⇒ -B
          ADD A          ; ve střádači je A-B
          JN ELSE        ; je-li (A-B)< 0, tak B je větší
THEN:     LOAD A         ; A je větší, tak A do střádače
          SAVE C         ; A ze střádače do C
          HALT           ; konec výpočtu
ELSE:     LOAD B         ; B je větší, tak B do střádače
          SAVE C         ; B ze střádače do C
          HALT           ; konec výpočtu
A         DEC 56         ; definice hodnoty A (zvoleno 56)
B         DEC -11        ; definice hodnoty B (zvoleno -11)
C         RESB 1         ; rezervování místa pro výsledek
          END START      ; konec programu s uvedením startu
```

Symboly uvedené vlevo před symbolickou instrukcí/direktivou se nazývají návěští/jména a reprezentují adresu této instrukce nebo adresu konstanty definované direktivou. Návěští se formálně odlišuje od jména tím, že je zakončeno dvojtečkou.

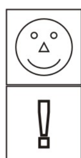
Pokud by měl náš překladač pracovat přímo, skončí s překladem neúspěšně již na první symbolické instrukci, protože neví, jakou adresu reprezentuje symbol A:

**Program v symbolickém jazyku:** **Překlad (program ve strojovém jazyku):**

| | | **Adresa paměti** | **Obsah paměti** . |
|---|---|---|---|
| | ORG 5 | 00101 | -------- |
| START: | LOAD A | 00101 | 001????? |



Proto překladač musí procházet zdrojový text dvakrát. Při prvním průchodu kontroluje syntaktickou správnost symbolických instrukcí, vytváří tabulku použitých symbolů a přiřazuje těmto symbolům adresy, které reprezentují. Pokud je pak v tabulce symbolů přiřazena všem symbolům jedinečná adresa, provede v druhém průchodu vlastní překlad.



Pozn.: Syntax je soubor pravidel pro tvorbu symbolických instrukcí a direktiv akceptovaných daným překladačem.

Překlad – první průchod (vytvoření tabulky symbolů):

| Symbol | Adresa | |
|---|---|---|
| START | 5 | 00101 |
| A | 22 | 10110 |
| ANEG | 10 | 01010 |
| B | 23 | 10111 |
| THEN | 16 | 10000 |
| TEST | 13 | 01101 |
| ELSE | 19 | 10011 |
| C | 24 | 11000 |



Překlad – druhý průchod (vlastní překlad):

**Program v symbolickém jazyku:** **Překlad (program ve strojovém jazyku):**

| | | **Adresa paměti** | **Obsah paměti** . |
|---|---|---|---|
| | ORG 5 | | |
| START: | LOAD A | 00101 | 00110110 |
| | JN ANEG | 00110 | 11001010 |
| | LOAD B | 00111 | 00110111 |
| | JN THEN | 01000 | 11010000 |
| | JMP TEST | 01001 | 10101101 |
| ANEG: | LOAD B | 01010 | 00110111 |
| | JN TEST | 01011 | 11001101 |
| | JMP ELSE | 01100 | 10110011 |
| TEST: | NEG | 01101 | 100xxxxx |
| | ADD A | 01110 | 01010110 |
| | JN ELSE | 01111 | 11010011 |
| THEN: | LOAD A | 10000 | 00110110 |
| | SAVE C | 10001 | 01111000 |
| | HALT | 10010 | 111xxxxx |
| ELSE: | LOAD B | 10011 | 00110111 |
| | SAVE C | 10100 | 01111000 |
| | HALT | 10101 | 111xxxxx |
| A | DEC 56 | 10110 | 00111000 |

| | | | |
|---|---|---|---|
| B | DEC -11 | 10111 | 11110101 |
| C | RESB 1 | 11000 | xxxxxxxx |
| | END START | | |

Poznamenejme, že program může být umístěn v paměti na „libovolném místě a že program může mít i jiný sled instrukcí – například data mohou být umístěna před instrukcemi:

| **Program v symbolickém jazyku:** | | **Překlad (program ve strojovém jazyku):** | |
|---|---|---|---|
| | | **Adresa paměti** | **Obsah paměti** |
| | ORG 0 | | |
| A | DEC 56 | 00000 | 00111000 |
| B | DEC –11 | 00001 | 11110101 |
| C | RESB 1 | 00010 | xxxxxxxx |
| START: | LOAD A | 00011 | 00100000 |
| | JN ANEG | 00100 | 11001000 |
| | LOAD B | 00101 | 00100001 |
| | JN THEN | 00110 | 11001110 |
| | JMP TEST | 00111 | 10101011 |
| ANEG: | LOAD B | 01000 | 00100001 |
| | JN TEST | 01001 | 11001011 |
| | JMP ELSE | 01010 | 10110000 |
| TEST: | NEG | 01011 | 100xxxxx |
| | ADD A | 01100 | 01000000 |
| | JN ELSE | 01101 | 11010000 |
| THEN: | LOAD A | 01110 | 00100000 |
| | JMP CONT | 01111 | 10110001 |
| ELSE: | LOAD B | 10000 | 00100001 |
| CONT: | SAVE C | 10001 | 01100010 |
| | HALT | 10010 | 111xxxxx |
| | END START | | |



**Hypotetický počítač č. 2**

Obsahuje hlavní paměť s kapacitou 256 slabik a procesor, který má 4 registry (R0, R1, R2 a SP) a pracuje s jednoslabikovými a dvouslabikovými instrukcemi. 3 registry (R0, R1, R2) jsou střádače, registr SP je registr ukazatele zásobníku (*Stack Pointer*), jehož funkce bude vysvětlena později. Formát jednoslabikových instrukcí je následující:

| Č. bitu | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | operační kód | | | | reg1 / reg | | reg2 / --- | |

a jejich význam je tento:

| **Operační Kód** | **Symbolický zápis** | **Popis operace** |
|---|---|---|
| 000 | NOP | prázdná operace (no operation) |
| 0001 | MOV reg1,reg2 | přenes operand z registru *reg2* do registru *reg1* |
| 0010 | MOV [reg1],reg2 | přenes operand z registru *reg2* do slabiky paměti, jejíž adresa je v registru *reg1* |
| 0011 | MOV reg1,[reg2] | přenes operand ze slabiky paměti, jejíž adresa je v registru *reg2*, do registru *reg1* |
| 0100 | CMP reg1,reg2 | porovnej čísla se znaménky, uložená v registrech *reg1* a *reg2* |

| | | |
|---|---|---|
| 0101 | PUSH reg | ulož obsah registru *reg* do zásobníku:<br>SP-1 → SP, *reg* → [SP] |
| 0110 | POP reg | naplň registr *reg* obsahem vrcholu zásobníku:<br>zásobníku [SP] → *reg*, SP+1 → SP |
| 0111 | HALT | stop |

Formát dvouslabikových instrukcí je následující:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| operační kód | | | | reg1/--- | | reg2/--- | | adresa *adr* / číslo *n* | | | | | | | |

první slabika — druhá slabika

Nechť jednotlivé instrukce provádí následující operace:

| **Operační Kód** | **Symbolický zápis** | **Popis operace** |
|---|---|---|
| 1000 | MOV reg1,[adr] | přenes operand ze slabiky paměti, jejíž adresa je *adr* do registru *reg1* |
| 1001 | MOV reg1,adr | přenes do registru *reg1* adresu *adr* |
| 1010 | MOV reg1,n | přenes do registru *reg1* číslo *n* |
| 1011 | CALL adr | skoč do procedury začínající na adrese *adr*<br>SP-1 → SP, IP → [SP], *adr* → IP |
| 1100 | RET n | vrať řízení volajícímu programu (návrat z procedury) a odstraň n slabik ze zásobníku<br>[SP] → IP, SP+n+1 → SP |
| 1101 | JMP adr | skoč na adresu *adr* |
| 1110 | JG adr | skoč na adresu *adr*, pokud při předchozím porovnání byl první operand větší než druhý operand (čísla se znaménky!) |
| 1111 | MOV reg1,[reg2+n] | přenes operand ze slabiky paměti, jejíž adresa je dána součtem obsahu registru *reg2* a čísla *n*, do registru *reg1* |



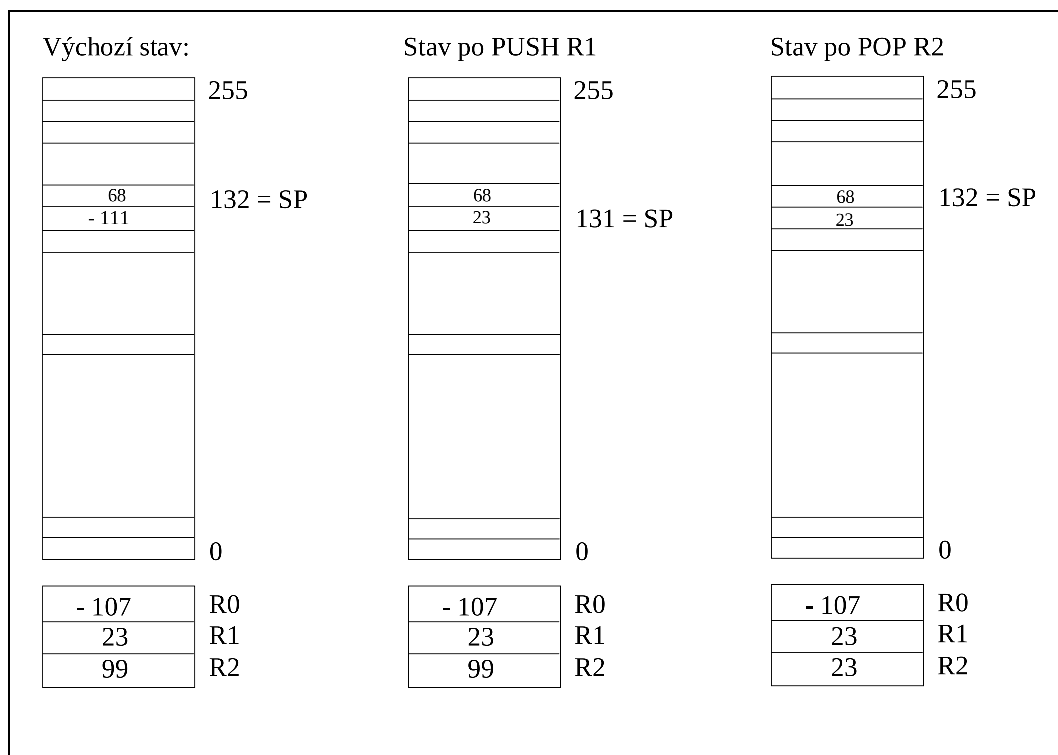
Na Obr. 3.2 je ukázán princip práce se zásobníkem. V levém sloupci obrázku je uveden zvolený výchozí stav paměti (256 slabik s adresami 0…255) a registrů procesoru (registr ukazatele zásobníku SP obsahuje hodnotu 132).

Instrukce PUSH R1 způsobí dekrementaci hodnoty ukazatele zásobníku SP (na hodnotu 131) a uloží do slabiky paměti odkazované touto hodnotou obsah registru R1 (tj. hodnotu 23).

Instrukce POP R2 uloží hodnotu uloženou v paměti na adrese dané aktuálním obsahem ukazatele zásobníku do registru R2 (tj. hodnotu 23) a inkrementuje obsah ukazatele zásobníku (jeho nová hodnota je 132). Poznamenejme, že obsah slabiky s adresou 131 se nemění (hodnota 23).

Ukazatel zásobníku ukazuje vždy na posledně uloženou hodnotu. Protože zásobník používá horní část paměti, nastavuje se počáteční hodnota ukazatele zásobníku na nulu. První položka ukládaná do zásobníku se pak uloží na adresu o jedničku nižší, tj. pro výše uvedenou paměť na adresu $(00000 - 00001)_2 = 11111_2 = 255_{10}$ .

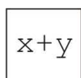





Obr. 3.2 Ukázka práce se zásobníkem

Definujme ještě následující direktivy:

| | |
|---|---|
| ORG adr | definice první adresy programu |
| END adr | označení konce programu a definice startovací adresy |
| DEC number | definice dekadické konstanty |
| RESB n | reservace n slabik paměti |
| name MACRO par | začátek definice makra *name* s formálními parametry *par* |
| ... | tělo makra |
| MEND | označení konce definice makra |



**Příklad:** E = *max*(A,B,C,D)

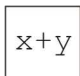
a) jednoduché řešení s opakováním posloupnosti instrukcí:

| **adresa paměti** | **symbolická instrukce nebo direktiva** | | **poznámka** |
|---|---|---|---|
| | | ORG 0 | |
| 00 | A | DEC 56 | ; definice konstant |
| 01 | B | DEC -11 | |
| 02 | C | DEC 21 | |
| 03 | D | DEC 13 | |
| 04 | E | RESB 1 | |
| 05 | POM1 | RESB 1 | |
| 06 | POM2 | RESB 1 | |
| 07 | START: | | ; start výpočtu |
| | | MOV R1,[A] | ; porovnání čísel A a B |
| 09 | | MOV R2,[B] | |
| 0B | | MOV R0,POM1 | |
| 0D | | CMP R1,R2 | |
| 0E | | JG VETSI_R1a | |
| 10 | | MOV [R0],R2 | |

```
11          JMP DALE_a
13      VETSI_R1a:
            MOV [R0],R1
14      DALE_a:                 ; větší z čísel do POM1
            MOV R1,[POM1]       ; porovnání čísel POM1 a C
16          MOV R2,[C]
18          MOV R0,POM2
1A          CMP R1,R2
1B          JG VETSI_R1b
1D          MOV [R0],R2
1E          JMP DALE_b
20      VETSI_R1b:
            MOV [R0],R1
21      DALE_b:                 ; větší z čísel do POM2
            MOV R1,[POM2]       ; porovnání čísel POM2 a D
23          MOV R2,[D]
25          MOV R0,E
27          CMP R1,R2
28          JG VETSI_R1c
2A          MOV [R0],R2
2B          JMP DALE_c
2D      VETSI_R1c:
            MOV [R0],R1
2E      DALE_c:                 ; větší z čísel do E
        `   HALT
            END START           ; START → IP
```



V programu se třikrát opakuje stejná posloupnost instrukcí, i když pokaždé s jinými operandy. Tato skutečnost je typická, a proto procesory nabízí její řešení pomocí procedur (obvykle s využitím zásobníku) a symbolické jazyky umožňují její řešení pomocí tzv. maker (makroinstrukcí).



b) řešení pomocí procedury:

**adresa paměti** | **symbolická instrukce nebo direktiva** | **poznámka**

```
            ORG 0
00      A   DEC 56              ; definice konstant
01      B   DEC -11
02      C   DEC 21
03      D   DEC 13
04      E   RESB 1
05      POM1 RESB 1
06      POM2 RESB 1
; procedura MAX předpokládá následující obsah zásobníku:
07      MAX: MOV R0,[SP+1]
09          MOV R2,[SP+2]
0B          MOV R1,[SP+3]
0D          CMP R1,R2
0E          JG VETSI_R1
10          MOV [R0],R2
11          JMP KONEC
13      VETSI_R1:
            MOV [R0],R1
14      KONEC:
            RET 3
16      START:                  ; start výpočtu
            MOV SP,0            ; nastavení ukazatele
                                ; zásobníku
```

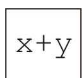
```
18          MOV R0,[A]        ; změny obsahu zásobníku:
1A          PUSH R0
1B          MOV R0,[B]
1D          PUSH R0
1E          MOV R0,POM1
20          PUSH R0
21          CALL MAX
23          MOV R0,[C]
25          PUSH R0
26          MOV R0,[POM1]
28          PUSH R0
29          MOV R0,POM2
2B          PUSH R0
2C          CALL MAX
2E          MOV R0,[D]        ; jiná posloupnost instrukcí
30          MOV R1,[POM2]     ; vedoucí ke stejnému uložení
32          MOV R2,E          ; parametrů do zásobníku
34          PUSH R0
35          PUSH R1
36          PUSH R2
37          CALL MAX          ; volání procedury MAX
39          HALT
            END START         ; START → IP
```

| |
|---|
| |
| čísloA |
| číslo B |
| adresa POM1 |
| IP ← SP |
| |

I když je předcházející program s využitím procedury delší než program původní, obecně je tomu naopak. Především jsou těla procedur (tj. všechny instrukce od vstupu do procedury do jejího opuštění instrukcí *RET*) obvykle výrazně delší a také symbolické jazyky umožňují ukládat do zásobníku adresy paměti a jejich obsahy (tj. například místo dvojic instrukcí *MOV R0,X* a *PUSH R0*, resp. *MOV R0,[X]* a *PUSH R0* umožňují používat pouze instrukce *PUSH X*, resp. *PUSH [X]*).



Posloupnost instrukcí procedury je jedinečná a vstupuje se do ní při každém volání této procedury (instrukcí *CALL*). Instrukce *RET n* provede návrat do hlavního programu (tj. na instrukci následující za instrukcí *CALL*) a současně „odstraní“ *n* slabik ze zásobníku posunutím ukazatele tohoto zásobníku (v našem případě, kdy parametr je uložen na jedné slabice odstraní *n* parametrů).



c) řešení pomocí makra:

**symbolická instrukce nebo direktiva** **poznámka**

```
        ORG 0
; definice makra MAX
MAX     MACRO P1,P2,P3            ; počátek definice makra
        LOCAL VETSI_R1,KONEC      ; definice lokálních návěští
        MOV R1,[P1]
        MOV R2,[P2]
        MOV R0,P3
        CMP R1,R2
        JG VETSI_R1
        MOV [R0],R2
        JMP KONEC
VETSI_R1:
        MOV [R0],R1
KONEC:
        MEND                      ; konec definice makra
A       DEC 56                    ; definice konstant
B       DEC -11
C       DEC 21
```

```
D      DEC 13
E      RESB 1
POM1   RESB 1
POM2   RESB 1
START:                      ; start výpočtu
       MAX A,B,POM1         ; volání makra MAX
       MAX C,POM1,POM2      ; volání makra MAX
       MAX D,POM2,E         ; volání makra MAX
       HALT
       END START            ; START → IP
```

Překlad výše uvedeného programu:

**adresa paměti** **symbolická instrukce nebo direktiva** **poznámka**

```
                    ORG 0
00          A       DEC 56
01          B       DEC -11
02          C       DEC 21
03          D       DEC 13
04          E       RESB 1
05          POM1    RESB 1
06          POM2    RESB 1
07          START:
                    MOV R1,[A]
09                  MOV R2,[B]
0B                  MOV R0,POM1
0D                  CMP R1,R2
0E                  JG @001
10                  MOV [R0],R2
11                  JMP @002
13          @001:   MOV [R0],R1
14          @002:
                    MOV R1,[C]
16                  MOV R2,[POM1]
18                  MOV R0,POM2
1A                  CMP R1,R2
1B                  JG @003
1D                  MOV [R0],R2
1E                  JMP @004
20          @003:   MOV [R0],R1
21          @004:
                    MOV R1,[D]
23                  MOV R2,[POM2]
25                  MOV R0,E
27                  CMP R1,R2
28                  JG @005
2A                  MOV [R0],R2
2B                  JMP @006
2D          @005:   MOV [R0],R1
2E          @006:
                    HALT
                    END START        ; START ⇒ IP
```



Posloupnost instrukcí, uvedenou v těle makra při jeho definici (makrodefinici), vygeneruje překladač při každém volání tohoto makra. Formální parametry přitom nahradí skutečnými parametry - makro se tedy „volá“ uvedením jména makra z makrodefinice, za kterým jsou uvedené skutečné parametry.



Definice lokálních návěští v makrodefinici způsobí, že při každém volání makra jsou tato návěští generována ve strojovém kódu jako jedinečná - v opačném případě by došlo k vícenásobnému generování stejného symbolu (zastupoval by více adres) a výsledný program by byl proto nepřeložitelný.



Obecně je program používající makra delší, na druhé straně však mírně rychlejší než program používající procedury. Rozhodnutí, který způsob při tvorbě programu použít, leží pouze na programátorovi a nelze dát obecný návod. Samozřejmě je vždy možné oba způsoby kombinovat.



Úkoly:

Pro hypotetický počítač napište program, který:
- nalezne maximální číslo v poli několika čísel bez znaménka
- nalezne minimální číslo v poli několika čísel se znaménkem
- zjistí, zda pole čísel bez znaménka tvoří monotónní posloupnost

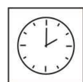
## 3.3 Shrnutí



Z informací uvedených v této kapitole je nutné pro úspěšné absolvování předmětu znát základní podsystémy počítače a princip jeho činnosti, dále pojmy *strojová instrukce*, *symbolická instrukce*, *direktiva* a princip činnosti asembleru - dvouprůchodového překladu z jazyka symbolických instrukcí do strojového jazyka. Také je nezbytně nutné správně pochopit princip práce se zásobníkem a programování procedur a maker.

# 4 ZÁKLADNÍ CHARAKTERISTIKY PROCESORŮ INTEL PENTIUM





15 hod

Cílem této kapitoly je poskytnout přehledové informace o procesorech firmy Intel (architektury IA-32) a dále základní informace o procesorech Pentium, které jsou nezbytně nutné pro pochopení práce těchto procesorů v základním režimu: informace o základních registrech, o typech celočíselných operandů, se kterými tyto procesory standardně pracují, o formátech instrukcí, které práci procesorů Pentium řídí, o způsobech adresování hlavní paměti a o principu přerušení a jejich obsluhy.

## 4.1 Procesory Intel (architektury IA-32):

**4.1 Procesory Intel**

Předmět IAS je zaměřen výhradně na základní režim procesorů Intel Pentium. Tento režim je emulován i v chráněném režimu (jde pak o tzv. virtuální režim), a protože procesory Pentium jsou použity ve většině výukových počítačů FIT VUT v Brně, je tak možné si veškeré teoretické poznatky bezprostředně prakticky ověřovat.



Stručný přehled dosavadního vývoje procesorů firmy Intel (architektury označované jako IA-32) je uveden v následující tabulce:

| Procesor | Datum zavedení | Taktovací kmitočet | Počet tranzistorů | Velikost registrů [b] | Adresovatel-ná paměť |
|---|---|---|---|---|---|
| 8086/8088 | 1978/1979 | 5-10 MHz | 29 K | GP: 16 | 1 MB |
| 80186/80188 | 1982 | 10-12 MHz | ?? | GP: 16 | 1 MB |
| 80286 | 1982 | 6-12 MHz | 134 K | GP: 16 | 16 MB |
| 80386 | 1985 | 16-33 MHz | 275 K | GP: 32 | 4 GB |
| 80486 | 1989 | 25-50 MHz | 1.2 M | GP: 32<br>FPU: 80 | 4 GB |
| Pentium | 1993 | 60 MHz | 3.1 M | GP: 32<br>FPU: 80 | 4 GB |
| Pentium Pro | 1995 | 200 MHz | 5.5 M | GP: 32<br>FPU: 80 | 64 GB |
| Pentium II | 1997 | 266 MHz | 7.0 M | GP: 32<br>FPU: 80<br>MMX: 64 | 64 GB |
| Pentium III | 1999 | 500 MHz | 8.2 M | GP: 32<br>FPU: 80<br>MMX: 64<br>XMM: 128 | 64 GB |
| Pentium 4 | 2000<br>2004 | 1.5 GHz<br>3.4 GHz | 42 M<br>125 M | GP: 32<br>FPU: 80<br>MMX: 64<br>XMM: 128 | 64 GB |
| Xeon | 2001<br>2002 | 1.7 GHz<br>2.2 GHz | 42 M<br>55 M | GP: 32<br>FPU: 80<br>MMX: 64<br>XMM: 128 | 64 GB |
| Pentium M | 2003<br>2004 | 1.6 GHz<br>2.1 GHz | 77 M<br>140 M | GP: 32<br>FPU: 80<br>MMX: 64<br>XMM: 128 | 4 GB |

| | | | | | |
|---|---|---|---|---|---|
| 64-bit Xeon | 2004<br>2005 | 3.6 GHz<br>3.3 GHz | 125 M<br>675 M | GP: 32, 64<br>FPU: 80<br>MMX: 64<br>XMM: 128 | 64 GB<br>1 TB |
| Pentium Extreme Edition | 2005 | 3.2 GHz | 230 M | GP: 32, 64<br>FPU: 80<br>MMX: 64<br>XMM: 128 | 64 GB |



Registry uvedené v tabulce mají následující význam:

- GP (***G**eneral **P**urpose*)
- FPU (***F**loating **P**oint **U**nit*)
- MMX (***M**ulti**M**edia E**X**tensions registers*)
- XMM (*Floating point E**X**tension for **M**ulti**M**edia*)



Pozn.: Registry prvních dvou skupin (GP, FPU) budou podrobně popsány v následujícím textu. S registry MMX, včetně základního popisu MMX instrukcí, které s těmito registry pracují, se zájemci mohou blíže seznámit ve volitelném předmětu PAS (Pokročilé Asemblery). Výklad XMM registrů a skupin instrukcí SSE, SSE2 a SSE3 (***S**treaming **SIMD** (Single Instruction Multiple Data) **E**xtension*), které tyto registry používají, přesahuje rámec obou předmětů a zájemce odkazujeme na stránky firmy Intel http://www.intel.com/.

U procesorů Pentium se rozlišuje pět režimů jejich možné činnosti:



1. Chráněný režim (*Protected mode*) - standardní pracovní režim počínaje procesory 80286.
2. Režim reálných adres (*Real address mode*) - základní režim. V tomto režimu se všechny procesory nachází po zapnutí nebo po přijetí signálu RESET a tento režim byl jediným možným pracovním režimem procesorů 8086.
3. Režim správy systému (*System Management Mode* – SMM) – speciální režim umožňující transparentní správu systému s ohledem na jeho bezpečnost.
4. Slučitelný režim (*Compatibility mode*) - umožňuje přímé spuštění 16-ti a 32 bitových aplikací pod 64 bitovým operačním systémem.
5. 64 bitový režim – pracuje s 64 bitovými aplikacemi pod 64 bitovým operačním systémem.



Pozn.: V předmětu IAS se budeme zabývat výhradně Základním režimem. Základní informace o chráněném režimu budou poskytnuty v předmětu PAS, popis zbývajících režimů přesahuje rámec obou předmětů.

S režimy činnosti úzce souvisí správa hlavní paměti. Rozeznávají se následující tři různé paměťové modely.

1. Monotónní / neměnný paměťový model (*Flat memory model*). Z pohledu programu jde o jednolitý paměťový prostor, který se nazývá lineárním paměťovým prostorem (pro 32 bitový režim má tento prostor maximální velikost $2^{32}$ slabik). Adresy, které lineární paměťový prostor zpřístupňují, se nazývají lineárními adresami.
2. Segmentovaný paměťový model (*Segmented memory model*). Z hlediska programu je paměť tvořena skupinami paměťových segmentů. Program může adresovat až 16 k (přesně 16383) segmentů, z nichž každý může mít velikost až $2^{32}$ slabik (počínaje procesory Pentium Pro až $2^{36}$ slabik).



3. Paměťový model reálných adres (*Real address mode memory model*) byl použit u procesoru 8086 a u všech dalších procesorů je od té doby zachován kvůli kompatibilitě programů. Jde o specifickou implementaci předchozího segmentovaného paměťového modelu, kde maximální velikost segmentu je 64 kB a maximální lineární paměťový prostor má velikost $2^{20}$ slabik. Odlišně se také určují počáteční adresy segmentů.

V základním režimu (tj. režimu reálných adres) procesorů Pentium lze používat pouze paměťový model reálných adres. Proto pouze tento model bude podrobněji vysvětlen v následujícím textu.

### 4. 2 Základní registry procesorů Pentium

Základní registry procesorů Pentium představuje

- Osm obecných (GP) 32 bitových registrů: EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP
- Jeden 32 bitový „příznakový a řídící registr“ EFLAGS
- Jeden 32 bitový registr ukazatele instrukcí EIP
- Šest 16-ti bitových segmentových registrů: CS, DS, SS, ES, FS, GS



**Obecné registry** jsou sice skutečně „obecné“, tj. mohou obsahovat libovolné operandy, nicméně mají i některá předurčená použití:

| | | |
|---|---|---|
| EAX | (*Accumulator*) | střádač |
| EBX | (*Base*) | ukazatel na data v datovém segmentu |
| ECX | (*Counter*) | čítač řetězových a smyčkových operací |
| EDX | (*Data*) | rozšíření střádače, ukazatel na I/O zařízení |
| ESI | (*Source Index*) | ukazatel na zdrojová data řetězových operací, index |
| EDI | (*Destination Index*) | ukazatel na cílová data řetězových operací, index |
| EBP | (*Base Pointer*) | ukazatel na data v zásobníkovém segmentu |
| ESP | (*Stack Pointer*) | ukazatel zásobníku |

Zdůrazněme, že především registr ESP se používá výhradně pro svůj předurčený účel a jiné jeho použití je „nebezpečné“ a nežádoucí!

Bity všech registrů jsou zásadně číslovány zprava počínaje nulou. Nejvyšší (tj. levý) bit u 32 bitových registrů má proto číslo 31.

U všech osmi obecných registrů je možné používat samostatně jejich dolní poloviny, které se pak označují jako AX, BX, CX, DX, SI, DI, BP, SP (obecné registry šestnáctibitových procesorů 8086 až 80286) a u prvních čtyř dokonce dvě dolní slabiky:

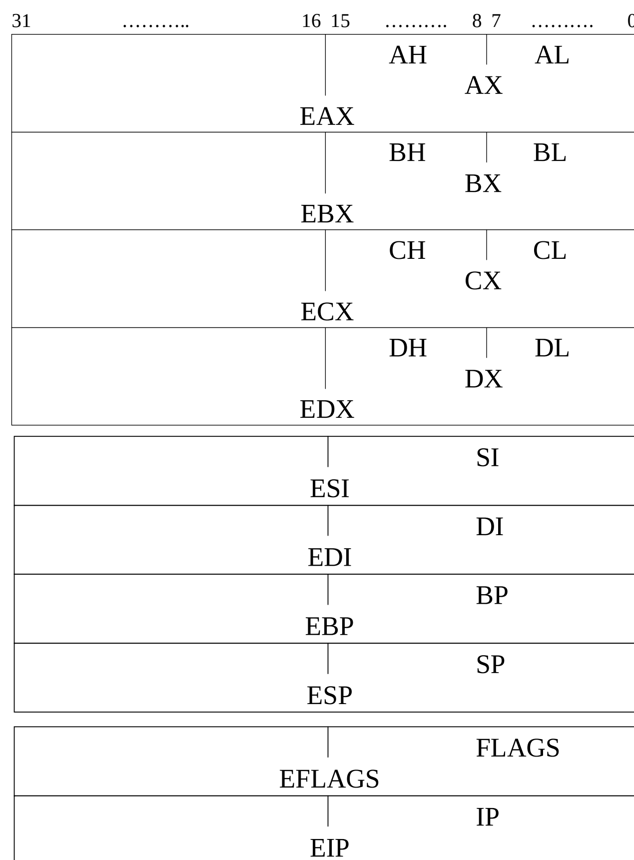
- bity č. 0 – 7: AL, BL, CL, DL (L … *Low*)
- bity č. 8 – 15: AH, BH, CH, DH (H … *High*)

V 16ti bitovém režimu se používají dolní poloviny registru EFLAGS (označuje se FLAGS) a registru EIP (označuje se IP). Situaci přehledně uvádí následující obrázek:



Pozn.: Písmeno E v názvu registrů znamená, že registry mají rozšířenou délku (***E**xtended*) – původní registry byly 16ti bitové!

Pozn.: V 64 bitových procesorech jsou základní 32 bitové registry rozšířeny na 64 bitů a jsou doplněny osmi novými 64 bitovými registry. Označují se jako RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP, RFLAGS, RIP, R8 až R15. U všech registrů se mohou samostatně použít jejich dolní poloviny, tj. registry EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP a R8D až R15D (D… *Doubleword*). U nových registrů je možné použít i jejich dolní slova R8W až R15W (W… *Word*) a dokonce i dolní slabiky R8L až R15L (L…*Low*).




Obr. 4.1 Základní 32 bitové registry procesorů Pentium



## Registr EFLAGS

Registr příznaků EFLAGS obsahuje příznaky stavové, řídící a systémové. Na následujícím obrázku jsou systémové příznaky, jejichž popis přesahuje rámec předmětu, označeny písmenem X, reservované bity jsou označeny přímo jejich hodnotami (nesmí se měnit) a jediný řídící příznak symbolem DF. Zbývající příznaky jsou stavové.

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X | X | X | 0 | X | X | OF | DF | X | X | SF | ZF | 0 | AF | 0 | PF | 1 | CF |

Obr. 4.2 Registr EFLAGS – Příznaky použitelné v základním režimu

**Význam řídícího a stavových příznaků:**

| | | |
|---|---|---|
| DF | *Direction Flag* | nastavuje směr operací u řetězových instrukcí |
| CF | *Carry Flag* | indikuje přenos/výpůjčku z/do nejvyššího bitu |
| PF | *Parity Flag* | indikuje sudou paritu v nejméně významné slabice výsledku (nastavuje se při sudém počtu jedniček) |
| AF | *Auxiliary Flag* | indikuje přenos/výpůjčku z/do bitu č. 3 |
| ZF | *Zero Flag* | indikuje nulový výsledek (při nulovém výsledku se nastaví na jedničku, jinak se nuluje) |
| SF | *Sign Flag* | indikuje záporný výsledek (při záporném výsledku se nastaví na jedničku, jinak se nuluje) |

| | | |
|---|---|---|
| OF | *Overflow Flag* | indikuje přetečení výsledku operací se znaménkovými čísly (při přetečení se nastaví na jedničku, jinak se nuluje) |



Pozn.: Při popisu jednotlivých instrukcí bude význam jednotlivých příznaků upřesněn.

Pozn.: Systémový příznak na bitech č. 13 a 12 je dvoubitový (tj. může nabývat čtyř různých hodnot).



**Registr EIP**

Registr EIP obsahuje 32 bitový ukazatel následující instrukce, tj. instrukce, která se bude jako další v pořadí provádět. V 16ti bitovém režimu obsahuje registr IP 16ti bitový ukazatel následující instrukce. Obsah registru EIP, resp. IP mohou měnit pouze skokové instrukce.



**Segmentové registry**

Segmentové registry jsou 16ti bitové registry, do kterých se ukládají selektory segmentů. Pomocí těchto selektorů se pak určují počáteční adresy odpovídajících paměťových segmentů – výpočet těchto adres je pak pro různé paměťové modely odlišný.

Dva segmentové registry mají speciální určení: CS (*Code Segment*) pro segment instrukcí a SS (*Stack Segment*) pro zásobníkový segment. Zbývající čtyři segmenty (DS, ES, FS a GS) se používají pro datové segmenty. Obsah registru CS mohou měnit pouze některé skokové instrukce, obsah ostatních segmentových registrů je možné měnit libovolně.

## 4. 3 Typy celočíselných operandů procesorů Pentium

Procesory Pentium mohou přímo pracovat s následujícími celočíselnými operandy:

- binárními čísly bez znaménka,
- binárními čísly se znaménkem,
- binárně kódovanými dekadickými čísly,
- slabikami,
- řetězy slabik, slov a dvouslov,
- řetězy bitů.

Pozn.: operandy, se kterými mohou pracovat FPU procesorů Pentium, budou popsány v kapitole 8.



**Binární čísla bez znaménka**

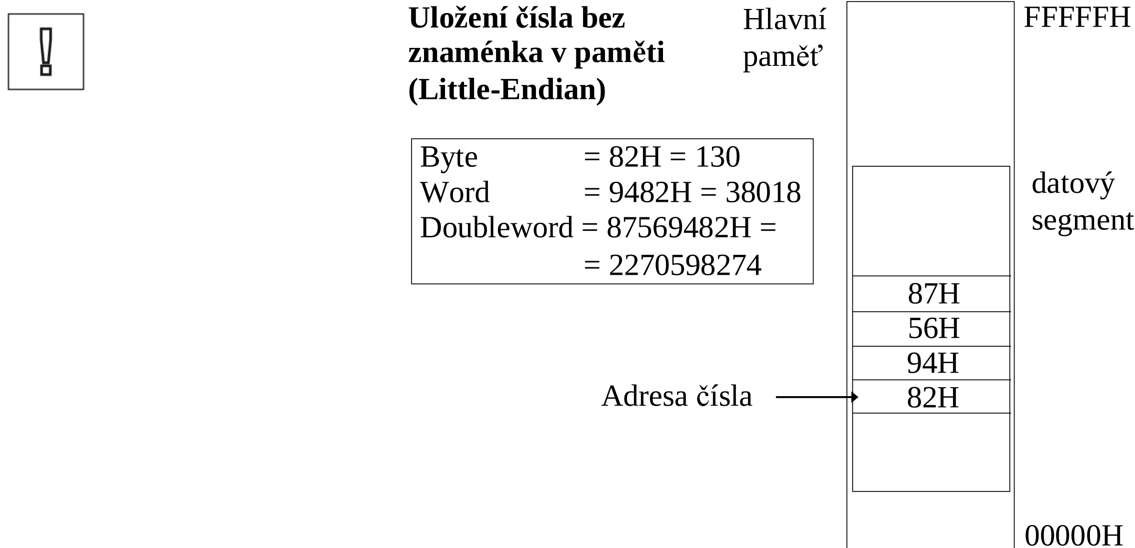
V základní režimu mohou procesory Pentium pracovat s bezznaménkovými čísly uloženými na slabikách, slovech a dvouslovech. Při operacích násobení a dělení mohou být v některých případech mezivýsledky dočasně uloženy na čtyřslovech.

| Počet bitů | Rozsah zobrazení | Přesnost zobrazení | Označení |
|---|---|---|---|
| 8 | <0, 255> | 2.4 | Byte (slabika) |
| 16 | <0, 65535> | 4.8 | Word (slovo) |
| 32 | <0, 4294967295> | 9.6 | Doubleword (dvouslovo) |
| 64 | $<0, 2^{64}-1>$ | 19.2 | Quadword (čtyřslovo) |

V registrech se všechna čísla ukládají v přirozeném tvaru, tj. nejvyšší bit je uložen vlevo (bit č. 7, resp. č. 15, resp. č. 31), nejnižší bit vpravo (bit č. 0).

Do paměti se bezznaménková čísla ukládají po slabikách. Nejnižší slabika se ukládá do adresované slabiky, vyšší slabiky se ukládají postupně do slabik s vyššími adresami. Tento způsob ukládání se nazývá „*little - endian*“ a je ukázán na Obr. 4.3.

Pozn.: Opačný způsob ukládání se nazývá „*big - endian*“.




Obr. 4.3 Uložení bezznaménkových čísel v paměti (*Little - Endian*)

**Binární čísla se znaménkem**

V základní režimu mohou procesory Pentium pracovat se znaménkovými čísly uloženými opět pouze na slabikách, slovech a dvouslovech. Stejně jako u bezznaménkových čísel mohou být v některých případech při operacích násobení a dělení mezivýsledky dočasně uloženy na čtyřslovech.

| Počet bitů | Rozsah zobrazení | Přesnost zobrazení | Označení |
|---|---|---|---|
| 8 | <−128, 127> | 2.1 | Byte (slabika) |
| 16 | <−32768, 32767> | 4.5 | Word (slovo) |
| 32 | <−2147483648, 2147483647> | 9.3 | Doubleword (dvouslovo) |
| 64 | <−$2^{63}$,$2^{63}$−1> | 19.0 | Quadword (čtyřslovo) |

V registrech se opět všechna čísla ukládají v přirozeném tvaru, tj. nejvyšší/znaménkový bit je uložen vlevo (bit č. 7, resp. č. 15, resp. č. 31), nejnižší významový bit vpravo (bit č. 0).

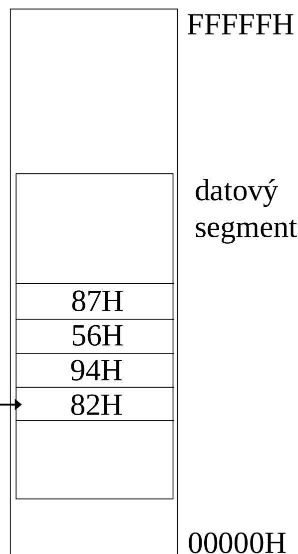
Do paměti se znaménková čísla ukládají stejně jako bezznaménková čísla po slabikách. Nejnižší slabika se ukládá do adresované slabiky, vyšší slabiky se ukládají postupně do slabik s vyššími adresami (Obr. 4.4).



**Uložení čísla se znaménkem v paměti (Little-Endian)**

| | |
|---|---|
| Byte | = 82H = −126 |
| Word | = 9482H = −27518 |
| Doubleword | = 87569482H = |
| | = −2024369022 |

Adresa čísla




Obr. 4.4 Uložení znaménkových čísel v paměti (*Little - Endian*)



**Binárně kódovaná dekadická čísla**

S BCD čísly pracuje skupina instrukcí dekadické aritmetiky – tato BCD čísla mohou být pouze jednomístná nebo dvoumístná (FPU umožňuje přímou práci s osmnáctimístnými BCD čísly se znaménkem! – viz kap. 8), pro práci s delšími BCD čísly je nutné vytvořit potřebné programy.

Jednomístná BCD čísla (BCD *integers*, *unpacked*) jsou představována jedinou BCD číslicí uloženou na dolních čtyřech bitech adresované slabiky. Pro operace sečítání a odečítání mohou mít horní čtyři bity této slabiky libovolné hodnoty, pro operace násobení a dělení musí být vynulované.

Dvoumístná BCD čísla (BCD *integers*, *packed*) jsou umístěna v adresované slabice; horní čtyři bity představují vyšší, dolní čtyři bity pak nižší BCD číslici.

BCD unpacked Integer:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| x | x | x | x | BCD | | | |

BCD packed Integer:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BCD | | | | BCD | | | |



**Slabiky**

Se slabikami pracuje pouze skupina instrukcí SETcc, které mohou obsahy adresovaných slabik buď vynulovat, nebo nastavit na hodnotu 1.



**Řetězy slabik, slov a dvouslov**

S řetězy slabik, slov a dvouslov pracuje skupina řetězových instrukcí. Řetěz představuje souvislá oblast slabik, slov nebo dvouslov v paměti, která začíná adresovanou slabikou (nejnižší slabikou slova nebo dvouslova). Délka řetězu je dána obsahem registru CX a směr řetězu od adresované slabiky (k horní nebo dolní části paměti) je určen hodnotou příznaku DF v registru EFLAGS. Bližší

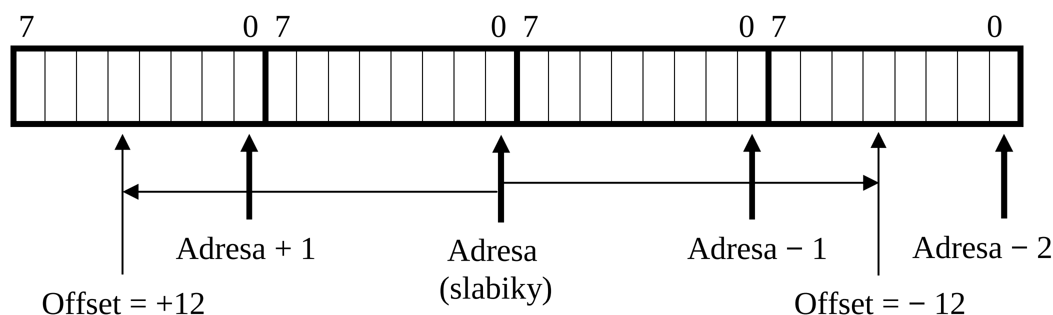
podrobnosti jsou uvedeny v popisu principů práce řetězových instrukcí (kap. 5.1).



**Řetězy bitů**

S řetězy bitů pracuje skupina instrukcí určená pro práci s bity. Adresa příslušného bitu je dána znaménkovým číslem – odsazením/offsetem od bitu č. 0 adresované slabiky.

V registru je offset vždy kladný a je dán přímo číslem bitu po vyčíslení podle vztahu: (zadaná_hodnota *mod* délka_registru). Adresování bitu v paměti ukazuje Obr. 4.5.




Obr. 4.5 Adresování bitů v paměti

**4.4 Formáty instrukcí a adresování operandů Procesorů Pentium**

Instrukce procesorů Pentium mají proměnnou délku od jedné do šestnácti slabik, a každou instrukci mohou předcházet teoreticky až čtyři jednoslabikové prefixy/předpony. Obecný formát instrukce procesorů Pentium je uveden na Obr. 4.6.

| Předpony | Operační kód | ModR/M | SIB | Posunutí | Operand |
|---|---|---|---|---|---|
| 0-4 slabiky | 1-2 slabiky | 0-1 slabika | 0-1 slabika | 0/1/2/4 slab. | 0/1/2/4 slab. |

Obr. 4.6 Formát instrukcí procesorů Pentium



**Předpony** ovlivňují práci instrukce, a přestože jejich použití není příliš časté, je nutné mít o nich alespoň základní znalosti.

V základním režimu pak přichází v úvahu použití následujících předpon (z každé skupiny vždy pouze maximálně jedna):

- předpona pro změnu velikosti operandu (66H)
- předpona pro změnu velikosti adresy (67H)
- předpona pro změnu segmentového registru:
    - CS (2EH)
    - SS (36H)
    - DS (3EH)
    - ES (26H)
    - FS (64H)
    - GS (65H)
- předpona pro řízení řetězových instrukcí
    - REP/REPE/REPZ (F3H)
    - REPNE/REPNZ (F2H)

Pořadí předpon, pokud je jich použito současně více, nemá na činnost instrukce vliv. Praktické použití jednotlivých předpon pak bude blíže popsáno v dalším textu.



**Operační kód** určuje operaci, která se provedením instrukce vykoná. Je uložen na jedné nebo dvou slabikách, u některých instrukcí používá ještě tři bity ve slabice ModR/M. Operační kód jednoznačně určuje, zda je či není následován slabikou ModR/M, a dále přímým operandem - o přítomnosti slabiky SIB a posunutí, pro které se často používá anglický název *displacement*, rozhoduje slabika ModR/M.

Pokud instrukce pracuje s operandy, určuje jeden z bitů operačního kódu, který však nemá pevnou pozici pro všechny instrukce, zda daná instrukce bude pracovat se slabikovými nebo delšími operandy. Základní režim předpokládá implicitně 16ti bitové operandy i 16ti bitové adresy, nicméně velikost obou těchto objektů lze pro příslušnou instrukci změnit na 32 bitů, pomocí výše uvedených předpon pro změnu velikosti operandu a pro změnu velikosti adresy. 64 bitové operandy, ani 64 bitové adresování v základním režimu použít nelze.



**Slabika ModR/M** je základní slabikou pro adresování operandů, se kterými instrukce pracuje. Její činnost je přitom bezprostředně ovlivněna danou velikostí operandu a danou velikostí adresy. Formát slabiky ModR/M je následující:

| 7 6 | 5 4 3 | 2 1 0 |
|---|---|---|
| Mod | Reg/Opcode | R/M |

Obr. 4.7 Formát slabiky ModR/M

Pole Mod určuje, zda operandem (nebo jedním ze dvou možných operandů) instrukce bude obsah paměti M (pro Mod = $00_2$, nebo Mod=$01_2$, nebo Mod=$10_2$), nebo obsah registru R (pro Mod = $11_2$). Adresa paměti/registru je pak blíže určena obsahem pole R/M podle následujících tabulek pro adresování registrů a paměti.



**Slabika SIB** (*Scale-Index-Base*) se používá pro adresování operandů ve 32 bitovém režimu - obsahuje-li pole R/M slabiky ModR/M hodnotu $100_2$, pak další obsah slabiky ModR/M je ignorován a výpočet efektivní adresy je určen slabikou SIB. Formát slabiky SIB je uveden na následujícím obrázku:

| 7 6 | 5 4 3 | 2 1 0 |
|---|---|---|
| Scale | Index | Base |

Obr. 4.8 Formát slabiky SIB

Význam jednotlivých polí slabiky SIB pro výpočet efektivní adresy je zřejmý z tabulek pro 32 bitové adresování.



**Posunutí (*displacement*)** je jednou částí tzv. efektivní adresy, jejíž výpočet je uveden v následujících tabulkách a jejíž použití k výpočtu skutečné adresy operandu v paměti je ukázáno dále.

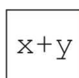


**Operand** uvedený v instrukci se nazývá přímým operandem. Může jím být znaménkové i bezznaménkové číslo nebo ordinální číslo znaku.

Pro adresy registrů je vyhrazeno tříbitové pole, které umožňuje adresovat osm různých registrů. S přihlédnutím k velikosti operandů pro danou instrukci, pak toto pole může adresovat 24 různých registrů podle následující tabulky:



| Adresa registru / Velikost operandu | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| 8 bitů | AL | CL | DL | BL | AH | CH | DH | BH |
| 16 bitů | AX | CX | DX | BX | SP | BP | SI | DI |
| 32 bitů | EAX | ECX | EDX | EBX | ESP | EBP | ESI | EDI |




Příklad: Adresa 6 = $110_2$ adresuje pro 8mi bitovou velikost operandu registr DH, pro 16ti bitovou velikost operandu registr SI a pro 32 bitovou velikost operandu registr ESI.

**Výpočet efektivní adresy operandu pro 16ti bitové adresování**:



| R/M | Mod = 00 | Mod = 01 | Mod = 10 |
|---|---|---|---|
| 000 | EA = BX + SI | EA = BX + SI + d8 | EA = BX + SI + d16 |
| 001 | EA = BX + DI | EA = BX + DI + d8 | EA = BX + DI + d16 |
| 010 | EA = BP + SI | EA = BP + SI + d8 | EA = BP + SI + d16 |
| 011 | EA = BP + DI | EA = BP + DI + d8 | EA = BP + DI + d16 |
| 100 | EA = SI | EA = SI + d8 | EA = SI + d16 |
| 101 | EA = DI | EA = DI + d8 | EA = DI + d16 |
| 110 | EA = d16 | EA = BP + d8 | EA = BP + d16 |
| 111 | EA = BX | EA = BX + d8 | EA = BX + d16 |

Z tabulky je zřejmý způsob tvorby efektivní adresy. Tato adresa je dána buď obsahy samostatných bázových nebo indexových registrů nebo jejich součty (Mod=00) s možným přičtením 8mi bitového d8 (Mod=01) nebo 16ti bitového d16 (Mod=10) posunutí. Jedinou výjimku z tohoto pravidla představuje kombinace Mod = 0 a R/M = 6, kdy je efektivní adresa dána přímo 16ti bitovým posunutím. Stručněji lze způsob tvorby efektivní adresy v 16 ti bitovém režimu vyjádřit takto:



$$EA = \begin{Bmatrix} BX \\ BP \end{Bmatrix} + \begin{Bmatrix} SI \\ DI \end{Bmatrix} + \begin{Bmatrix} d8 \\ d16 \end{Bmatrix},$$

přičemž alespoň jedna z uvedených tří částí musí být použita a je-li použita poslední část, musí být displacement 16ti bitový.

Důrazně upozorňujeme, že pro výpočet efektivní adresy nelze použít současně ani oba bázové registry, ani oba indexregistry a již vůbec ne jiné obecné registry!

Samostatné použití registru BP je překladačem/asemblerem překládáno jako

[BP + d8] s nulovým posunutím (d8 = 0).

Při výpočtu efektivní adresy se zanedbává případný přenos z nejvyššího bitu, proto přesnější zápis způsobu tvorby efektivní adresy je tento:



$$EA = \left( \begin{Bmatrix} BX \\ BP \end{Bmatrix} + \begin{Bmatrix} SI \\ DI \end{Bmatrix} + \begin{Bmatrix} d8 \\ d16 \end{Bmatrix} \right) \text{ mod } 2^{16}$$

Je tedy zřejmé, že efektivní adresa získaná výše popsaným způsobem je šestnáctibitová a umožňuje proto adresovat paměťový prostor 64 kB, kterému se říká **segment**.



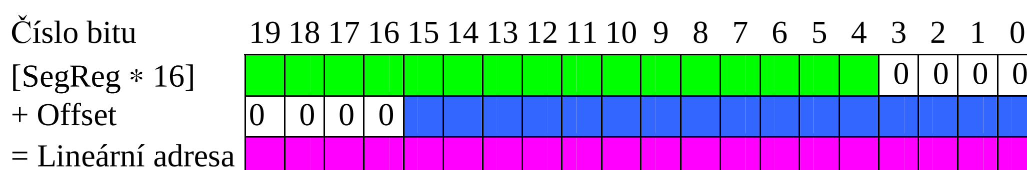
V základním režimu procesorů 8086 až 80286 byla skutečná/fyzická adresa 20ti bitová a umožňovala adresovat paměťový prostor o velikosti $2^{20}$B = 1MB. U pozdějších procesorů a tedy i u procesorů Pentium, se tato adresa nazývá lineární adresou. Pokud pak u těchto procesorů není zapnut stránkovací mechanizmus, jehož popis přesahuje rámec předmětu IAS, je fyzická adresa rovna adrese lineární.

K výpočtu lineární adresy se použije obsah segmentového registru, kterému se říká segmentová adresa, nebo také selektor. Selektor se vynásobí šestnácti (tj, posune se o čtyři bity doleva) a k němu se přičte efektivní adresa (obecně tzv. offset – viz dále), přičemž případný přenos se zanedbá:

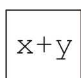


$$LA = ([SegReg * 16] + Offset) \text{ mod } 2^{20}$$

Dvojice SegReg a offset se nazývá úplným ukazatelem a zkráceně se zapisuje jako [SegReg:Offset]. Výpočet lineární adresy schématicky zachycuje následující obrázek:




Obr. 4.9 Výpočet lineární adresy v základním režimu (režimu reálných adres)




Příklad: Lineární adresa pro úplný ukazatel 4444:1234 = 45674h
Lineární adresa pro úplný ukazatel 4F1A:1B21 = 50CC1h



Vazba mezi segmentovými registry a offsetem je ukázána v následující tabulce:

| Typ odkazu paměti | Implicitní SegReg | Alternativní SegReg | Offset |
|---|---|---|---|
| Instrukce | CS | --- | IP |
| Implicitně zásobník | SS | --- | SP |
| Zdrojová řetězová data | DS | CS, ES, FS, GS, SS | SI |
| Cílová řetězová data | ES | --- | DI |
| Data adresovaná BP | SS | CS, DS, ES, FS, GS | EA |
| Jiná (standardní) data | DS | CS, ES, FS, GS, SS | EA |

Z tabulky jsou zřejmé všechny přípustné kombinace segmentových registrů a offsetů. Například lineární adresa instrukce je dána úplným ukazatelem CS:[IP] a nelze ji získat jiným způsobem, lineární adresa poslední položky uložené do zásobníku je dána úplným ukazatelem SS:[SP], opět bez možnosti změny, ale standardní data je možné odkazovat libovolným segmentovým registrem, přičemž offsetem je vždy efektivní adresa. Podrobnější vysvětlení ostatních odkazů bude uvedeno při popisu instrukcí v kapitole 5.

x+y

**Příklady výpočtu fyzických adres:**

Předpokládejme následující obsahy dále použitých registrů:
CS = 45F4h, SS = AB00h, DS = 5541h, ES = 1111h
IP = 2211h, SP = 1FE0h, BX = 3344h, BP = 2244h, SI = 1AD5h

Pak lineární adresy jsou následující:

- adresa instrukce: CS:IP = 45F40h + 02211h = 48151h
- adresa zásobníku: SS:SP = AB000h + 01FE0h = ACFE0h
- adresy operandů:
  - [BX] ⇒ DS:[BX] = 55410h + 03344h = 58754h
  - ES:[BX] ⇒ ES:[BX] = 11110h + 03344h = 14454h
  - [BP+4] ⇒ SS:[BP+4] = AB000h + 02244h + 4h = AD248h
  - [BX+SI+1] ⇒ DS: [BX+SI+1] = 55410h + 03344h + 01AD5h + 1h = = 5A22Ah

**Výpočet efektivní adresy operandu pro 32 bitové adresování**:

32 bitové adresování se v základním režimu prakticky výjimečně a vyhodnocená efektivní adresa přitom nesmí „ukazovat“ za hranici 64 kB segmentu!

| R/M <> 100 ⇒ slabika SIB v instrukci není | | | |
|---|---|---|---|
| R/M | Mod = 00 | Mod = 01 | Mod = 10 |
| 000 | EA = EAX | EA = EAX + d8 | EA = EAX + d32 |
| 001 | EA = ECX | EA = ECX + d8 | EA = ECX + d32 |
| 010 | EA = EDX | EA = EDX + d8 | EA = EDX + d32 |
| 011 | EA = EBX | EA = EBX + d8 | EA = EBX + d32 |
| 101 | EA = d32 | EA = EBP + d8 | EA = EBP + d32 |
| 110 | EA = ESI | EA = ESI + d8 | EA = ESI + d32 |
| 111 | EA = EDI | EA = EDI + d8 | EA = EDI + d32 |

| R/M = 100 ⇒ slabika SIB v instrukci je | | | |
|---|---|---|---|
| Base | Mod = 00 | Mod = 01 | Mod = 10 |
| 000 | EA = EAX+Index*s | EA = EAX+Index*s+d8 | EA = EAX +Index*s+d32 |
| 001 | EA = ECX+Index*s | EA = ECX+Index*s+d8 | EA = ECX+Index*s+d32 |
| 010 | EA = EDX+Index*s | EA = EDX+Index*s+d8 | EA = EDX+Index*s+d32 |
| 011 | EA = EBX+Index*s | EA = EBX+Index*s+d8 | EA = EBX+Index*s+d32 |
| 100 | EA = ESP+Index*s | EA = ESP+Index*s+d8 | EA = ESP+Index*s+d32 |
| 101 | EA = d32+Index*s | EA = EBP+Index*s+d8 | EA = EBP+Index*s+d32 |
| 110 | EA = ESI+Index*s | EA = ESI+Index*s+d8 | EA = ESI+Index*s+d32 |
| 111 | EA = EDI+Index*s | EA = EDI+Index*s+d8 | EA = EDI+Index*s+d32 |



Hodnoty *Base* a *Index* jsou dány obsahy stejnojmenných polí slabiky SIB (Obr. 4.8), hodnota *s* se určí z hodnoty pole *Scale* slabiky SIB podle následujícího vztahu:

$$s = 2^{Scale}$$

a může proto nabývat hodnot 1, 2, 4 a 8 (*Scale* může nabývat hodnot 0, 1, 2, 3).

Na rozdíl od 16ti bitového adresování je možné při 32 bitovém adresování používat všechny obecné registry a jejich kombinace - indexovým registrem však nesmí být registr ESP. Kód registru ESP ($100_2$) v poli *Index* slabiky SIB způsobí, že není uvažován žádný indexový registr! Další výjimkou je použití registru EBP: místo (EA = EBP) se musí použít (EA = EBP + d8, kdy d8 = 0) a místo (EA = EBP + Index ∗ s) se musí použít (EA = EBP + Index ∗ s + d8, kdy d8 = 0).

Vazba mezi segmentovými registry a offsetem při 32 bitovém adresování je ukázána v následující tabulce:



| Typ odkazu paměti | Implicitní SegReg | Alternativní SegReg | Offset |
|---|---|---|---|
| Instrukce | CS | --- | EIP |
| Implicitně zásobník | SS | --- | ESP |
| Zdrojová řetězová data | DS | CS, ES, FS, GS, SS | ESI |
| Cílová řetězová data | ES | --- | EDI |
| Data adresovaná bázovými registry EBP, ESP | SS | CS, DS, ES, FS, GS | EA |
| Jiná (standardní) data | DS | CS, ES, FS, GS, SS | EA |

Offset ve 32 bitovém adresování je sice skutečně uložen na 32 bitech, jeho hodnota však nesmí ukazovat za hranice segmentu (65535 pro slabiku, 65534 pro slovo, atd.), tzn. že nenulové bity offsetu mohou být pouze ve spodním 16ti bitovém slově. Lineární adresa se pak počítá dříve ukázaným způsobem (procesor je stále v základním režimu!):



$$LA = ([SegReg * 16] + Offset) \bmod 2^{20}$$

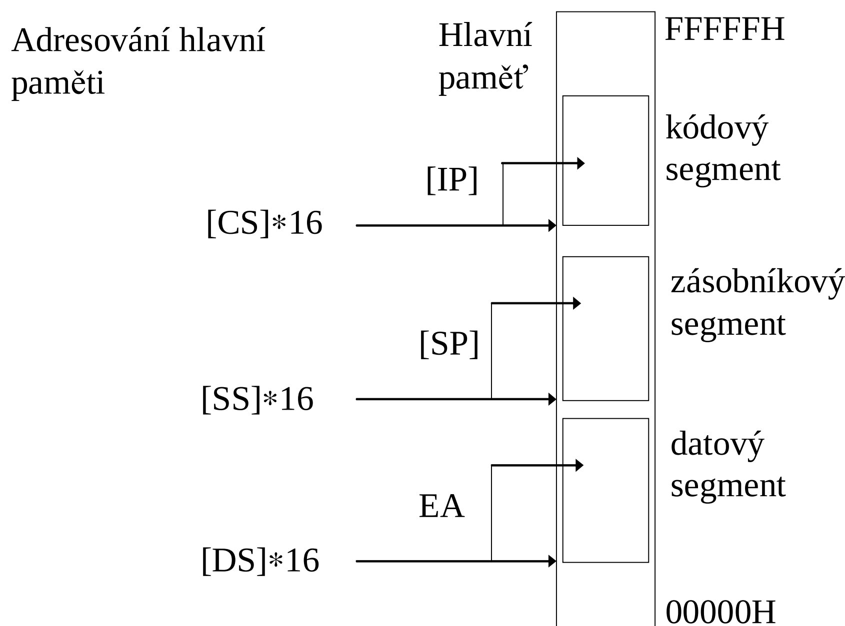
Porovnání 16ti bitového a 32 bitového adresování ukazuje následující tabulka:



| | 16ti bitové adresování | 32 bitové adresování |
|---|---|---|
| Bázové registry | BX, BP | Všechny obecné registry |
| Indexové registry | DI, SI | Všechny obecné registry kromě ESP |
| Displacement | 0, 8, 16 bitů | 0, 8, 32 bitů |
| Násobící konstanta | Není přípustná | 1, 2, 4, 8 |

Příklad standardního adresování hlavní paměti v základním režimu je uveden na následujícím obrázku:






Obr. 4.10 Příklad adresování hlavní paměti v základním režimu

**4. 5 Přerušení v základním režimu procesorů Pentium**



Procesory Pentium umožňují přerušit vykonávání programu v reakci na některou předem specifikovanou událost, vykonat tzv. obslužný program této události, a pak opět vrátit řízení přerušenému programu a pokračovat v jeho vykonávání. Události, které mohou přerušit vykonávání programu se dělí na

- Výjimky (*Exceptions*) - výjimkami jsou synchronní události generované procesorem při detekci jistých podmínek během zpracovávání instrukce (například pokus o dělení nulou).
- Přerušení (*Interrupts*) - přerušeními jsou asynchronní události generované obvykle vstupními a výstupními zařízeními (žádosti o obsluhu).

V základním režimu je pro uložení vstupních adres (úplných ukazatelů Segment:Offset) obslužných programů jednotlivých přerušení vyhrazen nejnižší 1kB hlavní paměti, tzv. tabulka přerušení. Protože každý ukazatel je uložen na čtyřech slabikách, může být v tabulce přerušení umístěno až 256 ukazatelů, které se nazývají vektory přerušení. Lineární/fyzická adresa nejnižší slabiky každého vektoru přerušení se snadno získá vynásobením čísla příslušného přerušení čtyřmi.

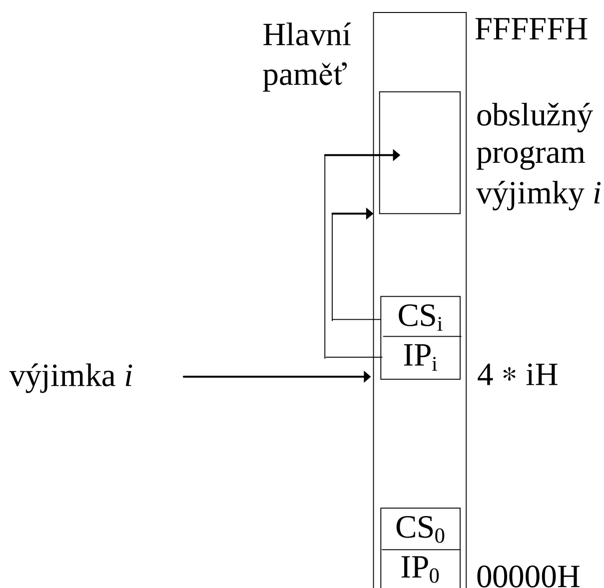
Procesor reaguje na oba typy událostí, tj. výjimky i přerušení, prakticky stejným způsobem.

V základním režimu uloží do zásobníku spodní slovo registru EFLAGS (bity č. 15 až 0), poté vynuluje v tomto registru některé systémové příznaky, do zásobníku uloží obsah registru CS a nakonec obsah registru IP – poslední dva registry obsahují adresu (úplný ukazatel) instrukce, která se bude vykonávat po návratu do přerušeného programu a které se říká návratová adresa. Poté se obsahy registrů CS a IP naplní obsahy příslušného vektoru přerušení a tím je vykonán skok do obslužného programu. Poznamenejme, že návrat z obslužného programu zabezpečí instrukce IRET.



Typický průběh přerušení a jeho obsluhy tedy vypadá takto (Obr. 4.10):

0. Vykonává se program.
1. Je generována výjimka číslo *i*, nebo vnější zařízení požaduje přerušení číslo *i*.
2. Dokončí se prováděná instrukce.
3. Do zásobníku se uloží obsah registrů EFLAGS[15:0], CS a IP.
4. Z tabulky přerušení se přečte adresa obslužného programu (*Interrupt Service Routine*, ISR) z lineární/fyzické adresy LA = FA = $i * 4$.
5. Adresa ISR se uloží do CS:IP.
6. Provádí se ISR.
7. Po ukončení ISR (instrukcí IRET) se vyjme ze zásobníku návratová adresa a uloží se do registrů IP a CS, a příznaky, které se uloží se do registru EFLAGS[15:0].
8. Pokračuje se ve vykonávání původního programu od místa, ve kterém bylo jeho vykonávání přerušeno.




Obr. 4.11 Příklad zpracování přerušení v základním režimu



Procesory Pentium rozeznávají čtyři druhy přerušení:

- výjimky (generované CPU)
- maskovatelné (INTR) vnější přerušení
- nemaskovatelné (NMI) vnější přerušení
- programové přerušení - libovolnou výjimku či přerušení lze vyvolat i instrukcemi INT n, kde *n* je číslo přerušení.



Výjimky generované CPU nelze maskovat, stejně jako vnější nemaskovatelné přerušení NMI – proto způsobí přerušení programu vždy. Ostatní vnější přerušení jsou zpracována vnějším řadičem přerušení, který do CPU posílá maskovatelný signál INTR, tzv. žádost o přerušení. Maskování (zákaz/povolení žádosti) přerušení lze ovládat systémovým příznakem IF (*Interrupt Flag*) v registru EFLAG.

V následující tabulce je pro informaci uveden přehled možných výjimek:

| **Číslo** | **Zdroj** | **Popis** |
|---|---|---|
| 0 | Instrukce DIV , IDIV | Chyba dělení |
| 1 | Odkaz paměti (instrukcí i dat) | Ladění |
| 2 | Nemaskovatelné vnější přerušení | NMI (*Non Maskable Interrupt*) |
| 3 | Instrukce INT3 | Bod přerušení (*Break-point*) |
| 4 | Instrukce INTO | Přetečení |
| 5 | Instrukce BOUND | Překročení hranice/rozsahu |
| 6 | Chybný kód, Instrukce UD2 | Chybný operační kód |
| 7 | Instrukce FPU a WAIT | Nedostupný koprocesor |
| 8 | Každá instrukce, NMI, INTR | Dvojitá chyba |
| 9 | | Rezervováno |
| 10 | Přepnutí úlohy | Chybný TSS (*Task State Segment*) |
| 11 | Instrukce pracující se seg. registry | Nepřítomný segment |
| 12 | Instrukce zásobníku a práce se SS | Chyba zásobníkového segmentu |
| 13 | Odkaz paměti nebo jiná kontrola | Obecná chyba |
| 14 | Odkaz paměti | Chyba stránky |
| 15 | | Rezervováno |
| 16 | Instrukce FPU a WAIT | Chyba FPU |
| 17 | Odkaz paměti | Kontrola (*Alignment check*) |
| 18 | Závislé na modelu procesoru | Kontrola (*Machine check*) |
| 19 | Instrukce SIMD FPU | Výjimka SIMD |
| 20-31 | | Rezervováno |
| 32-255 | Instrukce INT n, INTR | Maskovatelná vnější přerušení |



Úkoly:

- Vyjmenujte registry procesorů Pentium
- Vyjmenujte standardní celočíselné operandy procesorů Pentium
- Zvolte si náhodně obsahy všech registrů a spočítejte fyzické adresy:
    - Následující instrukce,
    - vrcholu zásobníku
    - Operandu se zvolenou kombinací bázového a indexového registru
- Popište princip přerušení a jeho obsluhy

## 4.3 Shrnutí



Informace uvedené v této kapitole jsou většinou zásadně důležité – výjimku tvoří pouze části popisující procesory Intel (kap. 4.1) a adresování ve 32 bitovém režimu (část kapitoly 4.4), jejichž znalost není nutná.

Nezbytně nutné je tedy znát všechny základní registry procesorů Pentium, typy standardních celočíselných operandů a formáty instrukcí těchto procesorů, všechny způsoby adresování operandů v základním 16ti bitovém režimu (složky efektivních a fyzických adres) a princip přerušení a jeho obsluhy.